

EXHIBIT 1

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.

Plaintiff,

v.

GOOGLE INC.

Defendant.

Case No. CV 10-03561 WHA

**OPENING EXPERT REPORT OF JOHN C. MITCHELL
REGARDING COPYRIGHT**

**SUBMITTED ON BEHALF OF PLAINTIFF
ORACLE AMERICA, INC.**

**CONFIDENTIAL PURSUANT TO PROTECTIVE ORDER
Highly Confidential Attorneys Eyes Only**

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	Retention	1
B.	Scope of Work Performed and Expected Testimony	1
C.	Compensation	3
II.	BACKGROUND AND QUALIFICATIONS	4
A.	General	4
B.	Copyright	6
III.	MATERIALS CONSIDERED AND RELIED ON	8
IV.	BACKGROUND: JAVA AND ANDROID	9
A.	The Java Platform: Technical Benefits and Consequences of Adopting the Java Programming Language	9
B.	The Java Class Libraries	13
C.	Success of Java and the Java Ecosystem	16
D.	Java Platform Inventions	19
E.	Google Android Products	20
F.	Android Market	23
G.	The Android Platform on End-User Devices	25
H.	Android Application Development	27
I.	Android's Adoption of the Java Platform	30
J.	Google's Decision to Adopt Java for Android and Why Alternatives Are Suboptimal	32
V.	THE COPYRIGHTS IN SUIT AND DETAILED OPINIONS	45
A.	Statement of Opinions and Findings	45
B.	Principles of Law	45
C.	Ownership	47
D.	Access	48
E.	Refresher on Object-Oriented Programming and the Java Language	49
F.	The Value of Application Program Interfaces	51
G.	Copyrightable Expression in the Java Platform	53
H.	The Android APIs Compared to the Java APIs	60
I.	Android Source Code Compared to the Java APIs	63
J.	Android Source Code Compared to Java Source Code	69

VI.	COPYRIGHTED FEATURES FORM BASIS FOR CUSTOMER DEMAND FOR ANDROID	75
A.	Google Understood that Applications Would Drive Consumer Demand for Android and Java Would Play a Central Role	75
B.	Google Recognized that Sun Microsystems's (now Oracle's) Copyrighted Features Would Be the Key to Attracting Millions of Application Developers to Android and OEMs.....	77
C.	Google Made Sun Microsystems's (now Oracle's) Copyrighted Features Necessary to Android.....	80
D.	Conclusion	81
VII.	CONCLUSION.....	82

I, John C. Mitchell, Ph.D., submit the following expert report (“Opening Copyright Report”) on behalf of plaintiff Oracle America, Inc. (“Oracle”):

I. INTRODUCTION

1. This report covers my review, analysis, and opinions regarding the copyrights asserted by Oracle against Google in the case known as *Oracle America, Inc. v. Google, Inc.*, pending in the United States District Court for the Northern District of California, Case No. 10-03561-WHA.

2. I will submit my review, analysis, and opinions regarding the patents-in-suit under separate cover on the due date agreed to by the Parties.

A. Retention

3. I have been retained to consult with counsel, review documents and other information, prepare expert reports, and be available to testify regarding my opinions on behalf of Oracle in connection with litigation brought by Oracle against Google.

B. Scope of Work Performed and Expected Testimony

4. In general I have reviewed materials to provide technical teaching and opinions regarding the patent and copyright infringement by Google in this case of the asserted claims of the patents and copyrights in suit. As stated in Oracle’s list of issues for expert testimony in Oracle’s case-in-chief, I expect to testify at trial regarding:

- Problem of application development and distribution for heterogeneous architectures;
- Origins, components, and versatility of the Java platform;
- The success of the Java platform, including technical accomplishments, attraction to developers, software development tools, and the Java ecosystem;
- Importance of compatibility to the Java platform;
- State of the art, subject matter, novelty, and significance of the claimed inventions in relation to the patents-in-suit;
- Priority dates of the patents-in-suit;
- Definition of a person of ordinary skill in the art for the patents-in-suit;

- Subject matter of the copyrights-in-suit, including the significance of Oracle's copyrighted works and the expressive choices available to Java and Android application programming interface (API) developers;
- Components of the Android platform;
- Components of the Android ecosystem;
- Modes of Google's distribution, control over Android source code, and likelihood of third parties modifying portions of Android alleged to infringe the intellectual property in suit;
- Infringement of the patents-in-suit, including direct and indirect infringement, and literal infringement and infringement under the doctrine of equivalents;
- Infringement of the copyrights-in-suit, including substantial similarity as between Java and Android APIs, Java APIs and Android source code, and Java source code and Android source code;
- Demand for and significance of the patents- and copyrights-in-suit to Android, including improved performance, improved security, widespread adoption, and speed to market;
- The absence of design-arounds and non-infringing alternatives to the Android technologies alleged to infringe the patents-in-suit and copyrights-in-suit;
- Whether the Android technologies alleged to infringe the patents-in-suit are specially made or adapted for use in infringement or have any substantial non-infringing uses;
- Performance benchmark tests, results, and analysis demonstrating the significance of the patents-in-suit to Android and the Java platform with respect to execution speed, memory savings, and architectural impact;
- Relevance of Oracle's Java patent portfolio to Android in the context of a hypothetical negotiation between Oracle and Google; and
- Google's acts of direct and indirect infringement of the patents and copyrights in suit.

I also expect to testify at trial with respect to matters addressed by experts testifying on behalf of Google. I may also testify on other matters relevant to this case, if asked by the Court or by the Parties' counsel.

5. I reserve the right to supplement or amend this report, if additional facts and information that affect my opinions become available. In particular, I understand that fact discovery closes on August 15, 2011. I have been informed that Google has yet to complete production of information that may affect my infringement analysis. Some information about

various versions of the accused software, systems, and applications is not publicly available. For example, I have not been able to examine the source code for Honeycomb, and I understand that Google plans to release future versions of Android software. None of these have been made available for my review. As such, my investigation into the specifics and extent of Google's infringement is ongoing. My report is based on the materials that have been available to me up to the date of this report.

6. My opinions are based on my education and my work experience as set forth in Exhibit A of this Report, the materials listed in Exhibit B of this Report, and the materials cited to in my Opening Copyright Report.

C. Compensation

7. I am being compensated for my work in connection with this matter at the rate of \$800 per hour. I have used this rate in the past for other cases.

8. My compensation is not conditioned on the outcome of this matter. Neither the amount of my compensation nor my hourly billing rate depends on whether I am obligated to testify at deposition or trial.

II. BACKGROUND AND QUALIFICATIONS

A. General

9. Information detailing my qualifications is included in my Curriculum Vitae, attached as Exhibit A.

10. I received my Ph.D. in Computer Science from MIT in 1984. I hold a Master's degree in Computer Science from MIT, and a Bachelors of Science in Mathematics with Distinction from Stanford University, which I received in 1982 and 1978 respectively.

11. My Ph.D. thesis was on topics related to the design and analysis of programming languages.

12. Between Stanford and MIT, I worked for two years as a research programmer for the University of Wisconsin.

13. Before I joined the faculty of Stanford, from 1984 until 1988, I was a Member of the Technical Staff at the Computing Science Research Center of AT&T Bell Laboratories.

14. I am currently the Mary and Gordon Crary Family Professor of Computer Science and (by courtesy) Electrical Engineering at Stanford University. From 1990 until 1997, I was an Associate Professor of Computer Science at Stanford University and from 1988 to 1990 I was an Assistant Professor of Computer Science.

15. At Stanford, my research has covered a variety of topics, including the design and analysis of programming languages, computer security, and mathematical logic. For example, I wrote research papers on type systems for object-oriented programming languages and developed principles that were adopted in the designs of the Java and .NET programming languages.

16. With my former Ph.D. student Stephen Freund, I wrote a series of papers studying properties of the Java bytecode verifier. With an additional former student, we designed, implemented and tested a form of Java generics, a concept that was subsequently added to the Java programming language.

17. My publications include three books on aspects of programming languages and over 140 research articles. A list of my publications is included in my Curriculum Vitae (Exhibit A).

18. I was elected as a Fellow of the Association for Computing Machinery in 2008. As explained on ACM's website: "The ACM Fellows Program was established by Council in 1993 to recognize and honor outstanding ACM members for their achievements in computer science and information technology and for their significant contributions to the mission of the ACM. The ACM Fellows serve as distinguished colleagues to whom the ACM and its members look for guidance and leadership as the world of information technology evolves." (*See* <http://fellows.acm.org/>.)

19. I am Editor-in-Chief of the Journal of Computer Security and I have served on the editorial board of ten other professional journals. I have also served as program committee chair or program committee member of many professional conferences on programming languages, computer security, and other areas of computer science.

20. The courses I have taught at Stanford include programming languages, programming language theory, computer and network security, web security, logic, and algorithms.

21. My research activities have been funded by U.S. Government agencies and gifts to Stanford from private companies. For example, I am currently the Principal Investigator (PI) of a project on secure information sharing that is funded by the Air Force Office of Scientific Research and the PI of a DARPA-funded project developing a programming language for computing on encrypted data. I am also a co-PI on the National Science Foundation Science and Technology Center called TRUST and a co-PI and Chief Computer Scientist of a project on security and privacy for healthcare information systems funded by the U.S. Department of Health and Human Services.

22. My research has also been partially funded over the last several years by gifts from private companies. While such gifts are often designated as intended to support a specific research project, they are gifts to Stanford University and not gifts to me personally.

23. In collaboration with Stanford faculty Dan Boneh and David Mazieres, I applied for a small research gift from Google to support research on “security for cloud clients with rich controlled sharing.” I believe that the final version of the application for funding was sent to Google by their deadline of October 15, 2010. Support for my research from this application has been placed on hold by Google for the duration of this litigation.

24. I am co-inventor of U.S. Patent No. 7,870,610, titled “Detection of Malicious Programs.”

25. I have served as an expert witness or consultant in connection with a number of litigations. A list of prior cases in which I have testified or consulted as an expert over the past years is included in my Curriculum Vitae (Exhibit A).

B. Copyright

26. With particular relevance to my copyright analysis conducted in this case, I previously assisted with a copyright dispute for which I analyzed and prepared an expert report covering copyrightable elements of user interfaces.

27. In my 30 years of experience with computer software, I have reviewed a large amount of source code written by different programmers. Moreover, I teach programming language classes and assign programming projects that are graded as part of the class. As a result, I am generally familiar with the variations of code expression that arise when a set of programmers (including students) are asked to solve a programming problem. I should note that even on exam questions, when I’ve tried to narrow the question to limit the set of possible correct answers, students usually find many ways to write source code to express solutions.

28. I am also familiar with tools used to detect plagiarism or source code copying in a university setting. I have served on a juror pool on Stanford’s Judicial Affairs, which handles plagiarism cases, including plagiarism in programming courses. In my interactions with

Stanford's Judicial Affairs Office, I've discussed principles for detecting and determining plagiarism and how they apply to computer programming assignments.

29. Like every other person with programming experience with whom I have discussed this, I personally find programming a creative experience, analogous to technical writing, for example. In both cases, although there is a purpose to be served, there are many ways to accomplish the goal, and a wide range of expressive choices in doing so.

III. MATERIALS CONSIDERED AND RELIED ON

30. In arriving at my opinions provided in this report, I have considered a number of different sources of information that are identified in attached Exhibit B and referenced in my Opening Copyright Report.

31. In particular, I have reviewed the copyrighted works asserted in this lawsuit; Android source code, videos, and documentation made publicly available and produced by Google during the course of discovery.

32. In support of my analysis and rendered opinions, I intend to rely on the summary and report of Marc Visnick comparing Sun Microsystems's (now Oracle's) JDK 1.5 to Android Froyo and Apache Harmony, submitted to Google with my Opening Copyright Report. Likewise, I intend to rely on the summary report of Alan Purdy comparing Sun Microsystems's (now Oracle's) Java library API specifications to Android's library API specifications, also submitted to Google with my Opening Copyright Report.

33. In addition to the materials specifically identified, I may provide further exhibits to be used as a summary of or support for my opinions.

IV. BACKGROUND: JAVA AND ANDROID

34. I believe a background and tutorial on Java and Android will aid the jury in understanding and appreciating the technology at issue in this case. I will cite to Google documents where appropriate.

35. There are several interesting features and components of Java that have marked its unprecedented technological success: the programming language, bytecode instruction set specification, virtual machine, execution paradigm and execution platform, programming tools, and class libraries and application programming interface (API) specifications.

36. To be clear on vocabulary, the Java development platform comprises the Java compiler, class libraries, development environment and associated tools, and the Java execution platform. The Java execution platform comprises the Java Virtual Machine and other components associated with processing and executing compiled Java source code, including executable class libraries and the class file format.

37. The Android platform incorporates various features of Java, as detailed below.

A. The Java Platform: Technical Benefits and Consequences of Adopting the Java Programming Language

38. The Java programming language was designed by James Gosling and others at Sun Microsystems in the early 1990s.

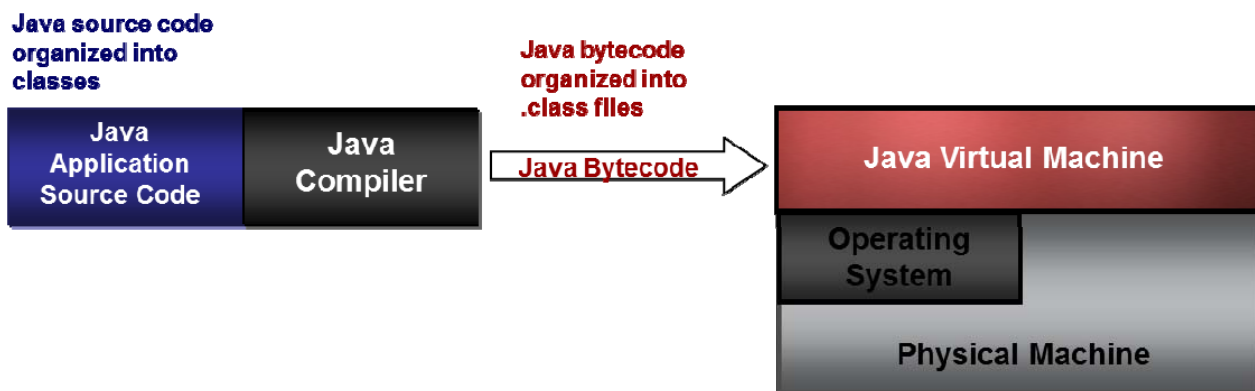
39. Java is an object-oriented language. In object-oriented languages, an *object* is a way of combining data and functions. For example, an Integer object would have an integer value, such as 3, together with functions on integers, such as addition and multiplication.

40. In most common object-oriented languages, programmers define *classes*. Each class provides a way for programs to create objects that have the functions defined in the class. For example, in an object-oriented program with an Integer class, a programmer can use this class to create several Integer objects in her program. In Java and other class-based object-oriented languages, all objects created from the same class have the same functions, also called

methods, but each object may have different data values. In Java, data values associated with an object are called *fields* of the object.

41. Object-oriented programming began in the 1960s and developed with the early programming language Simula. James Gosling credits the influence of Simula on Java in this statement that I quote in my programming languages textbook: “One of the most important influences on the design of Java was a much earlier language called Simula. It is the first OO language I ever used (on a CDC 6400!). ... [and] where the concept of a ‘class’ was invented.” In this statement, OO means “object-oriented.”

42. Java was developed and distributed with a novel execution platform. As documented in the original press release (*see* <http://web.archive.org/web/20080205101616/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml>), the Java execution platform allows software developers to write programs that, once compiled, may be transported and run on a variety of computers without re-compiling or rewriting them. This feature of Java is described as “write once, run anywhere.”



43. The key innovation of Java that supports “**write once, run anywhere**” is a combination of interrelated design decisions. First, Java source code programs are compiled into an intermediate executable form called *bytecode* that is stored in *class files*. Second, the virtual machine that is used to execute Java bytecode has specific functions that allow separately compiled class files to be incrementally loaded, verified, and then executed on any hardware platform that is equipped with a Java virtual machine. While the Java execution platform was

not the first to use bytecode and a virtual machine, I believe that it was the first to support separate transport and incremental loading, verification, and execution of bytecode files stored in a standardized portable format. Source code, compilers, bytecode, class files, and the Java loader, verifier and bytecode interpreter are described in the following paragraphs.

44. Since approximately 1960, computer programs have been written in human-readable programming languages, such as Fortran, Algol, C and C++. Code written in these human-readable languages, called *source code*, is not directly executable by computer hardware. Instead, it must be converted to machine code in some way. As the name suggests, *machine code* uses the set of instructions that are understood and executed by a specific hardware processor. Different computer chips may execute different sets of rudimentary machine code instructions.

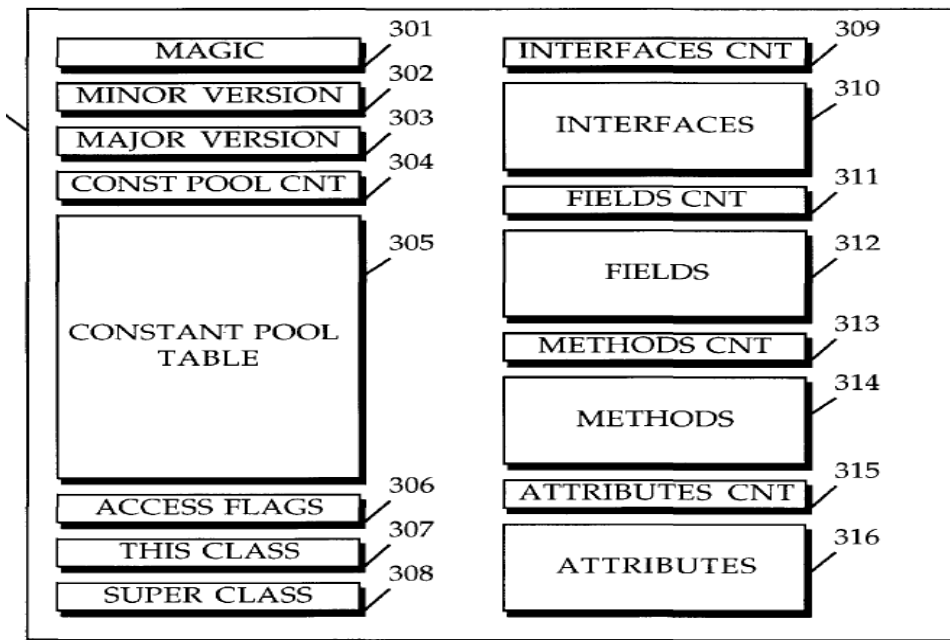
45. There are two traditional ways of converting source code into machine code. The first method uses a program called a compiler to convert an input file of source code to an output file of machine code that can be executed later. This machine code file can then be executed later, on a separate platform that does not have the source code or compiler. The second method uses a different kind of program called an interpreter. An interpreter processes source code instructions from an input file and executes corresponding machine instructions step-by-step, based on the source code instructions. Unlike a compiler, an interpreter continues to run as the source code program is interpreted. In other words, the source code and interpreter must be present on any platform where the program is interpreted.

46. Because a compiler converts source code to machine code for a specific machine, a traditional Apple Mac compiler, for example, produces compiled machine code that runs only on an Apple Mac and cannot be run on an IBM PC. (For many years, including early Java years, Apple used chips manufactured by Motorola and IBM PCs used incompatible chips manufactured by Intel.) In contrast, the same source code for an interpreted programming language can be run on different interpreters on different machines because each interpreter performs step-by-step execution using the appropriate machine instructions of the hardware it

runs on. However, it is not common to distribute software programs in their source code form. Source code represents the creative work of software designers and developers in a form that could expose a company's intellectual property to competitors.

47. The Java execution platform uses an intermediate form of executable code called bytecode, mentioned above. Bytecode instructions are not the machine instructions of any specific physical computer. Instead, they are the machine instructions of a *virtual machine*. Such virtual machine instructions are called bytecode apparently because the portion of the instruction that determines which operation is performed is often one byte long (*i.e.*, represented compactly as eight bits, each bit a 0 or 1).

48. The Java source code compiler transforms a source code file defining a Java class to a bytecode file that implements this class. The bytecode file representing a compiled class is called a Java *class file*. In addition to the specific bytecode instructions that implement the functions defined in the class, a Java class file has several other types of information, arranged in a very specific way. The Java class file format is defined in the Java Virtual Machine Specification (Lindholm and Yellin, Version 1, Version 2) and also described in U.S. Patent No. 5,966,702. The main sections of a Java class file are illustrated in Figure 3 of the '702 patent, reproduced below. In Java software development, source code is conventionally stored in files that have the ".java" extension, while class files produced by compiling source code to bytecode is conventionally stored in files that have the ".class" extension.



49. In the Java execution platform originally released in the mid-1990s, the virtual machine executes bytecode instructions using a program called a bytecode interpreter. In other words, bytecode instructions are executed, step-by-step, by an interpreter. Because bytecode interpreters were built for Apple Macs, IBM PCs, and Unix-based computers, the same bytecode file could be run on all of these different computers. In other words, once compiled to Java bytecode, the same program, written once, could be run anywhere.

50. The Java virtual machine has additional components, including a class loader and a bytecode verifier. The class loader locates class files as needed when a program executes and incrementally adds them to the execution environment. The bytecode verifier is a program that examines bytecode to check for certain errors and make sure it conforms to the rules of the Java programming language. The specific operation of the Java class loader and the Java bytecode verifier depend on the Java class file format.

B. The Java Class Libraries

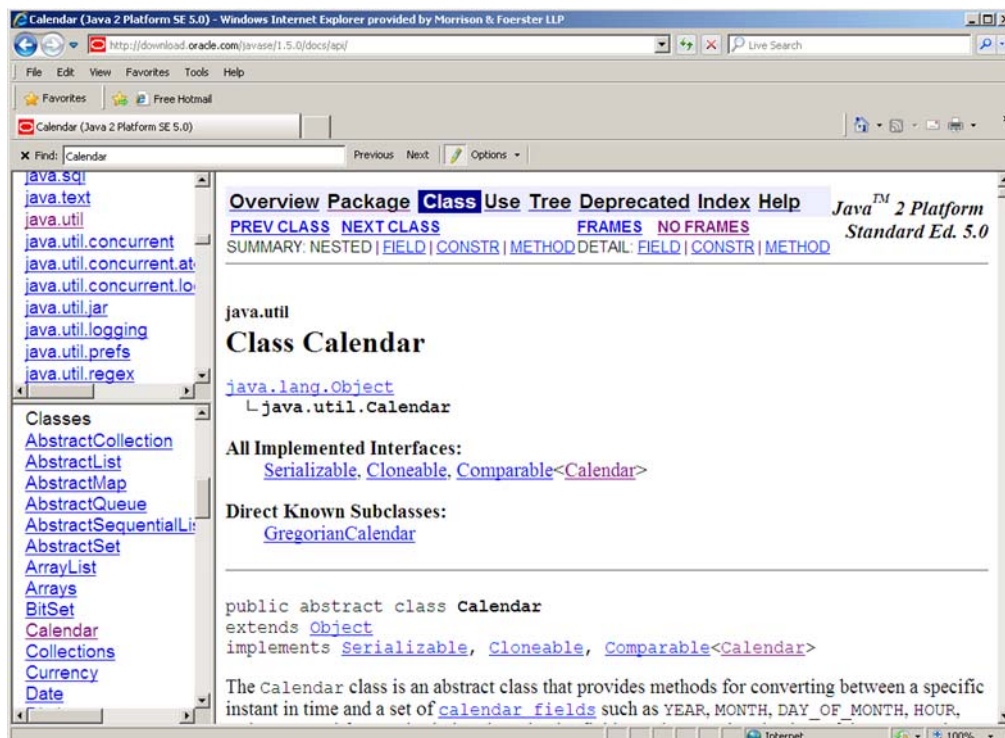
51. The Java programming language is also the gateway to extensive class libraries. The Java class libraries available in 1996, for example, were documented in a 1650-page book called *The Java™ Class Libraries: An Annotated Reference* (*see*

http://java.sun.com/docs/books/chanlee/first_edition/descript.html). These libraries included **core** packages with classes for input-output, networking, and other common functions. The libraries also included a Window Toolkit for programming user interfaces and Applet packages for building Java applets (a form of web-based Java program). The Java class libraries grew over time. These extensive libraries help programmers develop useful and substantial programs easily, without writing their own implementations of functions provided in the library. In other words, the Java class libraries speed and ease application development, and obviate the need for developers to program from scratch.

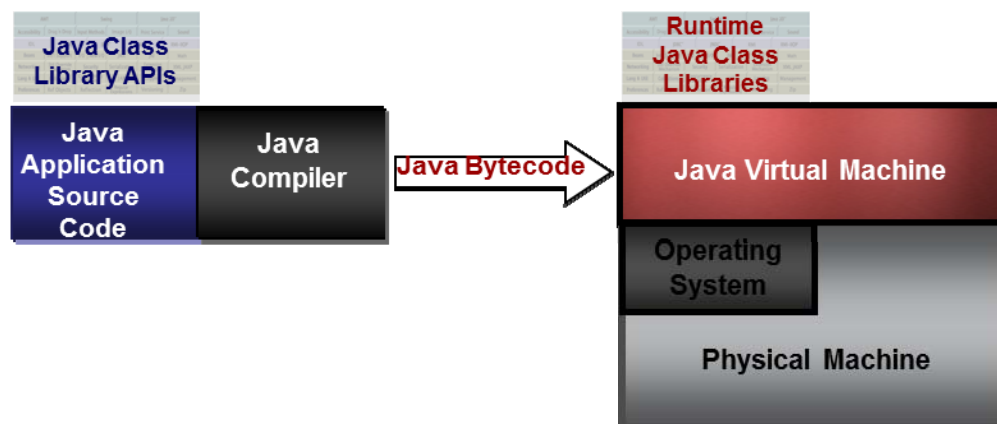
52. Generally, a software library provides a set of functions, classes, or other program entities that are designed to be used in a variety of programs. Once they are designed, built, and debugged, libraries make it easier to build new programs because the components provided by a library can be used directly without further programming effort. A programmer using a library writes code using what are commonly called Application Program Interfaces (APIs). An API consists of a set of names that can be used to access features of the library, together with specified conventions about their use. For example, an API allowing a program to determine the time of day might include a function called *time*, together with the convention that a call to this function returns an integer representing the clock time in a particular format.

53. A programmer using a software library does not need to have the source code for the library because the API tells the programmer how to use the library. However, a programmer using a software library must have access to an executable form of the library in order to build and run software that uses the library.

54. An example Java API from a current Java class library is shown in the screen shot below. This screen image shows a portion of the API for `java.util.Calendar`, the `Calendar` class from the `java.util` package. As stated in the text of the display window, the `Calendar` class provides methods related to time and calendar fields such as `YEAR`, `MONTH`, and `HOUR`.



55. Because Java programs run by loading and executing bytecode class files in the Java Virtual Machine, a program that uses library classes requires library class files to run. Therefore, the Java Development Kit (JDK) that programmers use to develop software contains or provides access to both Java class library APIs and associated bytecode class files (often in the form of .jar files that contain a number of class files). The relationship between library APIs and library class files used by a Java programmer is illustrated in the figure below.



C. Success of Java and the Java Ecosystem

56. The Java programming language and platform have been an outstanding success, by virtually any measure. In particular, Java has been one of a handful of leading programming languages for many kinds of applications over the past 15 years, rivaling C and C++ as the most widely used general purpose systems programming language, as discussed below.

57. Because it has been so successful, the Java language and platform brings with it a huge surrounding ecosystem of developers and tools, including system architects experienced in designing systems using Java, Java programmers and Q/A testers, Java development environments, Java development kits, Java program analysis and testing tools for improving the quality of Java software, instructional books and web sites, and related materials.

58. One measure of language popularity is reflected in the Tiobe Index, which is based on search engine results for a combination of search engines (see http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm). According to the following table based on this index, Java was among the top five programming languages in 1996, shortly after its release. Java has been among the top three languages since at least 2006. Additional information on the Tiobe web site shows Java as the top programming language since at least 2002, with a significant margin over both C and C++ at that time. (See <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.)

Programming Language	Position Jun 2011	Position Jun 2006	Position Jun 1996	Position Jun 1986
Java	1	1	5	-
C	2	2	1	1
C++	3	3	2	5
C#	4	8	-	-
PHP	5	5	-	-
(Visual) Basic	6	4	3	6
Objective-C	7	43	-	-
Python	8	7	26	-

Perl	9	6	6	-
Lua	10	50	-	-
Lisp	14	15	16	3
Ada	19	16	11	2

59. According to the Java History Timeline (see <http://www.oracle.com/technetwork/java/javase/overview/javahistory-timeline-198369.html>), for example, JDK 1.0 was released in 1996. One year later, in 1997, there were over 220,000 downloads of JDK 1.1 software in just three weeks, and over 2 million downloads by 1998. The 1997 JavaOne event drew 8,000 attendees, becoming the world's largest developer conference at that time. Two years later, attendance reached 20,000.

60. Java quickly became a popular teaching language. As a result, many colleges and universities taught Java to students, resulting in a large number of trained Java programmers. A report from a 1997 panel discussion at a computer science education conference concluded:

“[Java] has proven to be a powerful, general purpose, object-oriented language that fits very well with the concepts taught in many computer science courses, from beginning programming for all types of students to graphics to distributed programming. The ability to use it to create applets for algorithm animation provides support for alternative learning styles. Its platform independence is particularly useful with the mixture of equipment found among most departments, personnel and students. It is not without its problems, but the list of those problems does seem to be smaller than that for other languages, and some can be solved by using encapsulation to build a cleaner, higher level, and more easily used abstraction. Furthermore, students are motivated to learn Java as they perceive it to be fun and marketable, and this, in turn, makes the teacher's tasks more rewarding.”

(N.C. Schaller et al., Panel Report: Using Java in Computer Science Education, Proceedings of the 2nd Annual Conference on Integrating Technology into Computer Science Education, ITiCSE 1997, Uppsala, Sweden, 1-5 June, 1997. ACM 1997, ISBN 0-89791-923-8, retrieved from http://portal.acm.org/ft_gateway.cfm?id=266154&type=pdf.) The history of the ITiCSE Conferences on Innovation and Technology in Computer Science Education, which started in

1996 and continue to this day, is documented on the web (*see*

<http://www.cs.utexas.edu/users/csed/iticse/>).

61. The history of advanced placement exams, used in U.S. college admissions, also illustrates the prominence and widespread success of Java. The AP Computer Science (APCS) exam was introduced in 1984 when the language of instruction—both for APCS and for the vast majority of colleges and universities in the United States—was Pascal. (E. Roberts, “The Dream of a Common Language: The Search for Simplicity and Stability in Computer Science Education,” SIGCSE ‘04 Proceedings of the 35th SIGCSE technical symposium on Computer science education (2004), *available at* <http://www-cs-faculty.stanford.edu/~eroberts/papers/SIGCSE-2004/DreamOfACommonLanguage.pdf>.) While Pascal was widely used, the advancement of object-oriented programming and other factors lead to a revision of the exam after a decade. In 1995, the College Board announced that the APCS program would shift to C++ in the 1998-99 academic year, and the first C++ exams were administered in May 1999. (*Id.*) While C++ supports object-oriented programming, there are a number of reasons why teaching C++ to beginning programmers is cumbersome. As Eric Roberts, a national leader in computer science education and sometime-skeptic of Java explains, “Java, a new object-oriented programming language that had only just been introduced when the AP announced its C++ decision, quickly gained a foothold in the educational community and then began to blossom over the next few years. By the time the C++ exam was put in place, conventional wisdom held that Java, and not C++, was in the ascendant as the primary language for introductory computer science courses.” (*Id.*) As a result, the College Board announced in 2001 that the APCS program would move to Java beginning in 2004. (*Id.*) Dr. Roberts himself revised his influential 1995-1998 C++-based books and released Java-based textbooks in 2006 and 2008 (*see* <http://www-cs-faculty.stanford.edu/~eroberts/cv.html>).

62. In industry, Java has not only been important to Sun Microsystems and Oracle, but also to a number of other leading companies. For example, the importance of Java to IBM, one of the world’s largest computer companies over the past several decades, is documented in

<http://www-01.ibm.com/software/ebusiness/jstart/history.html>. For example, the JStart team worked to “evangelize and validate Java as a commercial development platform through both internal IBM and external industry clients.” The WebSphere Application Server developed from a Java-based research project (the Servlet Express Engine). IBM and another company, BEA, also collaborated on Java specifications in order to maintain compatibility between their competing products. (See, e.g., M. LaMonica, “IBM, BEA join on Java strategy,” CNET News (Nov. 25, 2003), available at <http://news.cnet.com/2100-7345-5111567.html>.) In addition to explaining the importance of Java to large companies that use it, the LaMonica article also explains the importance of platform compatibility to customers who buy Java software products.

63. A 2009 New York Times article explaining IBM’s interest in acquiring Sun Microsystems—before the acquisition by Oracle—explains that “I.B.M. uses Java extensively in its big software group, which trails only Microsoft in size” and notes that IBM “has its own Java-based tools for software developers, called Eclipse.” (S. Lohr and A. Vance, “I.B.M., Looking to Buy Sun, Sets Up a Software Strategy,” The New York Times Inside Technology article (Mar. 18, 2009), available at <http://www.nytimes.com/2009/03/19/technology/companies/19sun.html>.)

D. Java Platform Inventions

64. The Java language and platform contain many innovative ideas. Sun Microsystems filed many patents related to Java, including numerous patents on specific aspects of the Java compiler, the JDK, and the JVM.

65. Some representative inventions embodied in the Java platform are expressed in the seven patents asserted in this litigation. (I will discuss the patents in suit in my separate patent infringement report.)

66. In addition, the Java Class library source code and its APIs are subject to protection based on copyright law.

67. Since the Java Class library and its APIs are expressed in a specific and creative way, the source code and other aspects of this software ought to be accorded copyright

protection. The expression involved in both source code and the Java APIs is discussed later in this report.

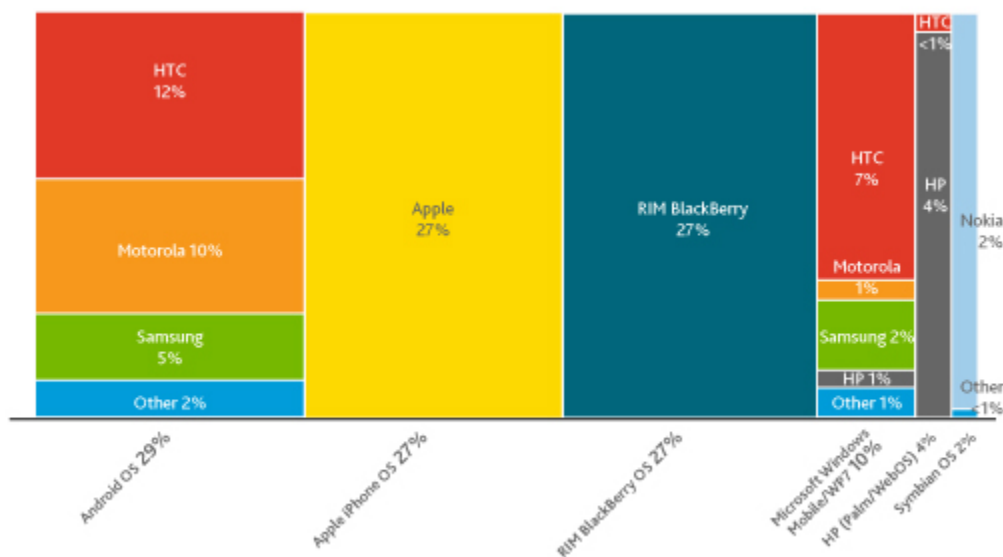
68. Beyond these exemplary inventions, there are many other creative, novel, and protected aspects of the Java platform. I will discuss this issue further in my patent infringement report.

E. Google Android Products

69. Google Android software typically runs on mobile devices such as smartphones and tablets. Unlike Apple iPhone and iPad software, which is intended for Apple-branded hardware, Google Android software is available on devices produced by a number of different manufacturers and sold under the manufacturer's brand names.

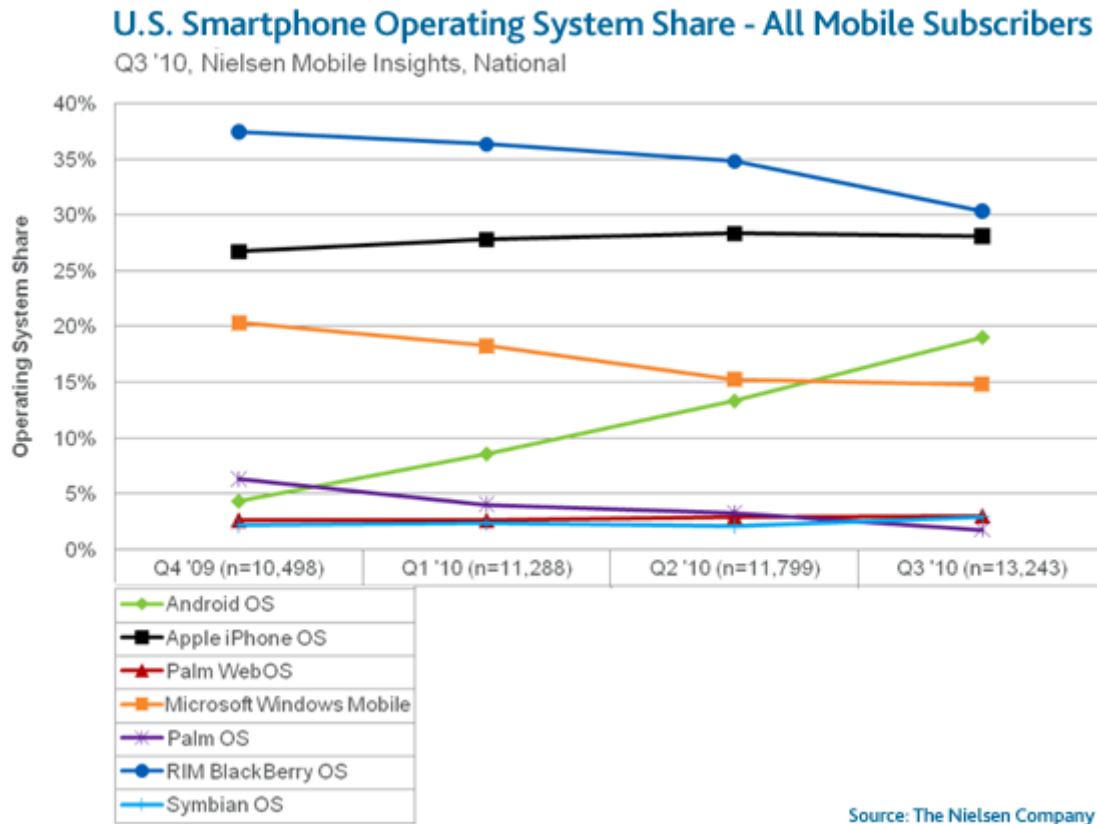
70. As smartphone software, Android has successfully achieved market share comparable to Apple iPhone, and RIM BlackBerry, each accounting for almost 30% of the current market. (*See, e.g.*, NielsenWire, "Who is Winning the U.S. Smartphone Battle?," Mar. 3, 2011, available at http://blog.nielsen.com/nielsenwire/online_mobile/who-is-winning-the-u-s-smartphone-battle/.)

71. The leading manufacturers of Android phones shown in the following market comparison are HTC, Motorola, and Samsung.



72. In 2008, the leading smartphones were Blackberry and iPhone. Over the course of that year, Blackberry rose from 4% penetration rate to 6%, while iPhone rose from 1% to 2%. (See, e.g., Nielsen Communication Trends, “ Highlights from the 2008 Convergence Audit and Consumer Electronics Monitor,” Dec. 2008, *available at* http://kr.en.nielsen.com/site/documents/CommunicationTrends_200810.pdf.) A year later, in 2009, approximately 15% of U.S. households owned a smart phone, with Blackberry reaching 8% market penetration and iPhone 4%. (See, e.g., Nielsen Communication Trends, “Highlights from the 2009 Nielsen Convergence Audit,” *available at* <http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/12/09-Nielsen-Convergence-Audit.pdf>.)

73. Android’s dramatic advance in 2010 is shown in the following graph (from <http://blog.nielsen.com/nielsenwire/wp-content/uploads/2010/11/smartphone-OS-share.png>). Clearly Android’s market share advanced rapidly in 2010, while most others were flat or declining. While there are many possible factors, consumer enthusiasm for phones with features common to iPhone and Android is evident.



74. The following features are highlighted on Google's Android "What is Android?" page (see <http://developer.android.com/guide/basics/what-is-android.html>):

- **"Application framework** enabling reuse and replacement of components
- **"Dalvik virtual machine** optimized for mobile devices
- **"Integrated browser** based on the open source WebKit engine
- **"Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **"SQLite** for structured data storage
- **"Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **"GSM Telephony** (hardware dependent)
- **"Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- **"Camera, GPS, compass, and accelerometer** (hardware dependent)
- **"Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE"

75. The main differences between Android and iPhone and competitors such as Blackberry and Windows Mobile appear to be the touchscreen, extensibility, and **a wide variety of user-installable applications, many developed by third party developers.** A developer

community sufficient to compete with the size and diversity of the iPhone applications (“apps”) appears critical to Android’s success.

76. Internal Google documents are consistent with this view (and more evidence will be discussed below):

BEGIN GOOGLE ATTORNEYS’ EYES ONLY

“There is no purpose of building an open platform other than to attract third-party developers to it. So anything that we would do to jeopardize the support of third-party developers would be bad for the success of the platform.” 4/5/2011 Rubin Dep. 90:7-91:23.

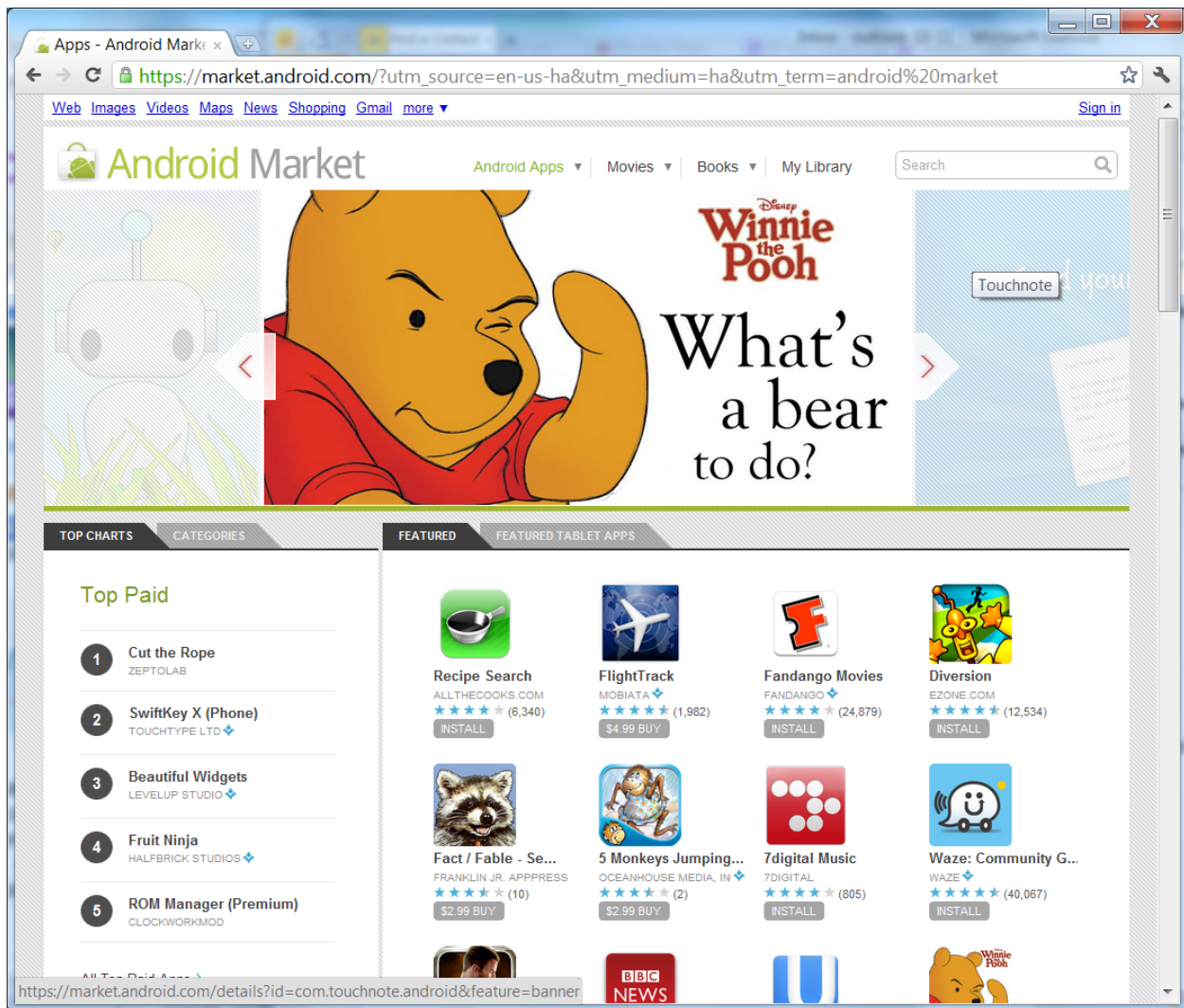
“Third-party developers contribute to the success of a platform by having their companies invest in the platform by basing their businesses on the platform. It was my intention to create an independent third-party developer ecosystem...” 4/5/2011 Rubin Dep. 24:6-25:11.

END GOOGLE ATTORNEYS’ EYES ONLY

F. Android Market

77. Android Market, accessible through a web browser or through an Android device, allows users to buy or download free Android Apps, books, and movies. As explained in help pages accessible from the Android Market web site, “Android Market offers quick, easy access to a wide variety of applications developed specifically for the Android platform. These have been created by developers all around the world, and have been rated by your fellow Android users.” (*See* <http://www.google.com/support/androidmarket/bin/answer.py?hl=en&answer=113407&topic=1100168>.)

78. From a developer’s point of view, Android Market provides a way for application developers to make applications available to Android users. A sample screen shot from the Android Market web site is shown below:



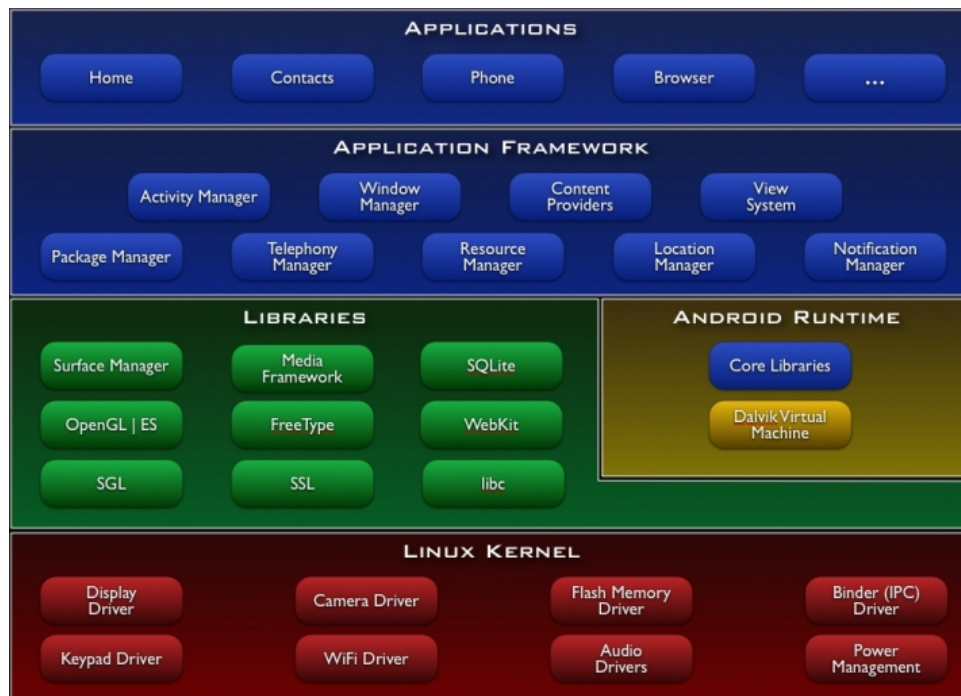
79. A tour of Android Market (*see, e.g.,* <http://www.businessinsider.com/how-to-use-the-new-android-market-2011-2>) shows that a user must sign in using a Google account before browsing Android apps by category, paid, free, or popularity. It is also possible to search for apps using a search bar and click on icons to read more information about any app. As the tour pages show, “You can check out reviews, ratings, screenshots, and videos of the app” and “If you want to download click ‘Install’ next to the app’s icon.”

80. Since a Google account is also a Gmail account, Android users appear to necessarily become Gmail users. When I entered my login information in order to use a phone and use Android Market to install an app, the phone was automatically configured to read my Gmail account on the phone.

G. The Android Platform on End-User Devices

81. In Google's own words, "Android is a software stack for mobile devices that includes an operating system, middleware and key applications." (See <http://developer.android.com/guide/basics/what-is-android.html>.)

82. A software stack involves interrelated software components that "stack" in the sense that they can be arranged into layers with software at higher layers depending on software at lower layers (and not conversely). (This concept is *not* related to the run-time stack used in program execution.) A Google diagram (from <http://developer.android.com/images/system-architecture.jpg>) illustrating major components of the system shows four layers consisting of (i) the Linux operating system, (ii) libraries and the Android runtime, (iii) an Application Framework, and (iv) Applications:



83. In the computer field generally, an *application* is a computer program that a user interacts with directly. According to this Android diagram, an Android application such as the browser relies on the application framework. The program that implements the browser can make method calls to software that is part of the application framework. The application framework in turn makes use of the Android runtime, which includes the "Core Libraries" and

“Dalvik Virtual Machine.” These also run under control of the Linux operating system and kernel.

84. Android uses a “multi-user Linux [operating] system in which each application is a different user.” (See <http://developer.android.com/guide/topics/fundamentals.html>.) This separates distinct applications from each other because different Linux users normally have different permissions. “By default, every application runs in its own Linux process” and “Each process has its own virtual machine (VM), so an application’s code runs in isolation from other applications.” (See <http://developer.android.com/guide/topics/fundamentals.html>.)

85. While Android applications are normally isolated from each other by the standard operating system mechanisms, “there are ways for an application to share data with other applications and for an application to access system services.” (See <http://developer.android.com/guide/topics/fundamentals.html>.) For example, two applications may “share the same Linux user ID, in which case they are able to access each other’s files.” (See <http://developer.android.com/guide/topics/fundamentals.html>.) Alternatively, “applications with the same user ID can also arrange to run in the same Linux process and share the same VM.” (See <http://developer.android.com/guide/topics/fundamentals.html>.)

86. Moreover, “An application can request permission to access device data such as the user’s contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.” (See <http://developer.android.com/guide/topics/fundamentals.html>.)

H. Android Application Development

87. The Android Developer Website (<http://developer.android.com>) provides extensive information about the Android platform, its components, and how they can be used by software developers.

88. The Google Android web site explains how Android applications are written in Java and compiled to produce .dex files that are executed by the Dalvik Virtual Machine.

Specifically, and as quoted directly from Google's Android Developer Website:

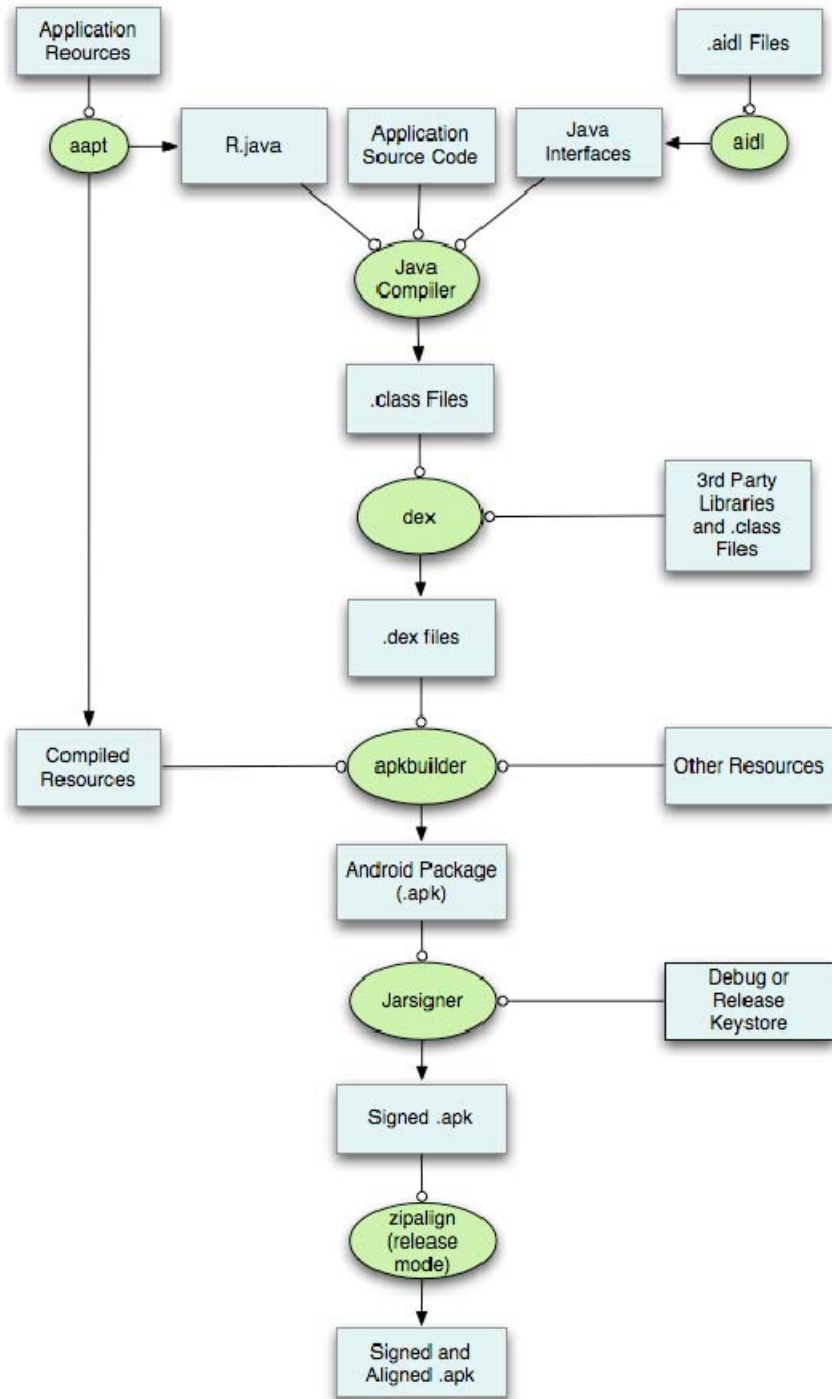
- “Android applications are written in the Java programming language” and compiled using tools in the Android SDK (software development kit). (See <http://developer.android.com/guide/topics/fundamentals.html>.)
- “Each Android application is compiled and packaged in a single file that includes all of the application's code (.dex files), resources, assets, and manifest file.” (See <http://developer.android.com/guide/appendix/glossary.html>.)
- “The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint.” (See <http://developer.android.com/guide/basics/what-is-android.html>.)
- “The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.” (See <http://developer.android.com/guide/basics/what-is-android.html>.)

89. Further information on the bytecode for the Dalvik VM is available at <http://source.android.com/tech/dalvik/dalvik-bytecode.html>; further information on the Dalvik Executable Format (.dex) is available at <http://source.android.com/tech/dalvik/dex-format.html>.

90. The Java Development Kit (JDK) provides the backbone of the Android SDK. In fact, the system requirements for installing the Android SDK include one of three operating systems, a Java Integrated Development Environment (IDE), and appropriate versions of the JDK.

91. For Eclipse, a popular IDE developed by IBM for Java, Google Android provides a specific Android Development Tools (ADT) plug-in. (See <http://developer.android.com/sdk/requirements.html>.) The ADT adds capabilities to Eclipse to let developers “quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug ... applications using the Android SDK tools, and

even export signed (or unsigned) .apk files in order to distribute ... application[s].” (See <http://developer.android.com/sdk/eclipse-adt.html>.) According to Android documentation, “Developing in Eclipse with ADT is highly recommended and is the fastest way to get started.” (See <http://developer.android.com/sdk/eclipse-adt.html>.)



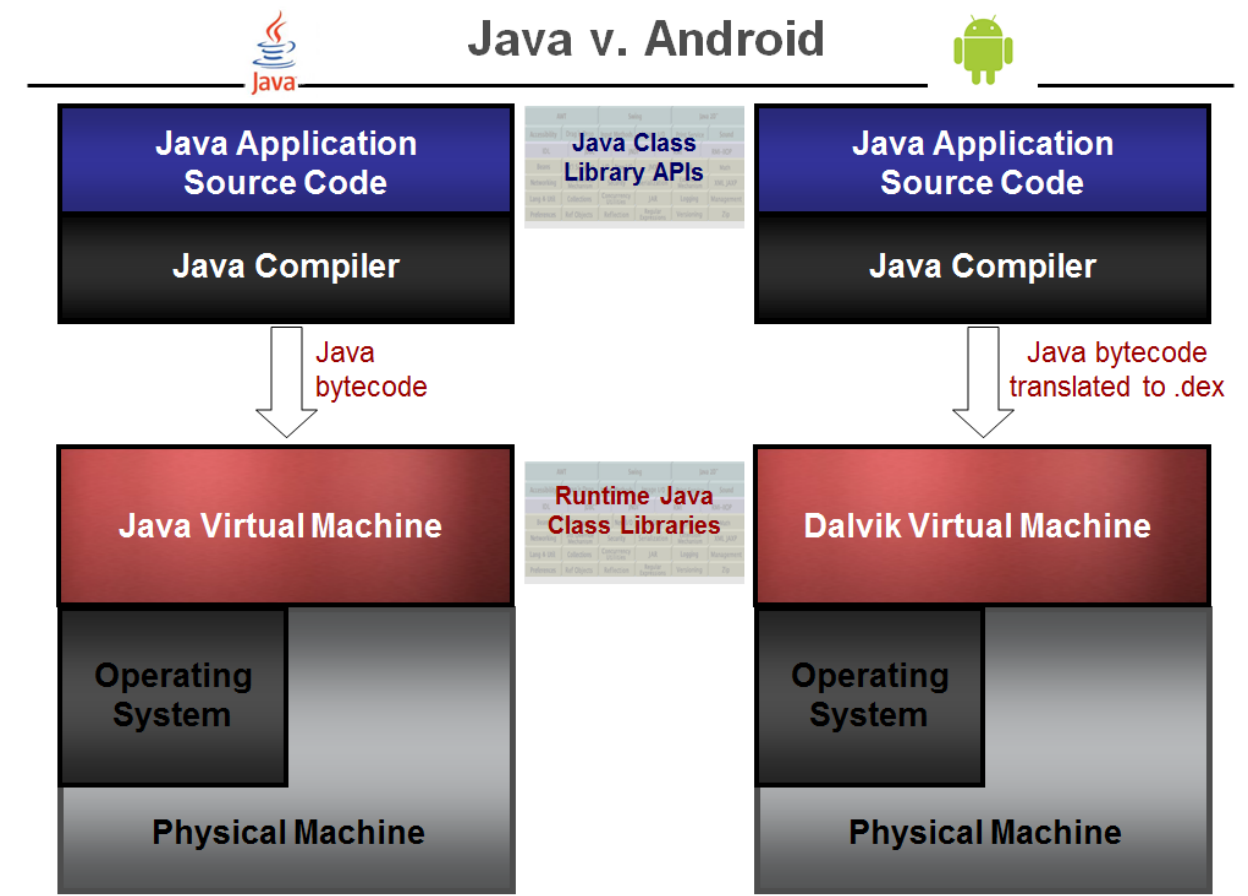
92. The main components used in building an Android application from source code are shown in the illustration immediately above (from <http://developer.android.com/guide/developing/building/index.html>). As this illustration shows, application source code is compiled using the Java programming language compiler to produce a .class file. This is the same process used for compiling java source code to .class files. The Android SDK then uses the *dx* tool to convert a .class file to the .dex format, and *apkbuilder* to produce an Android package (.apk) file. Android packaged files are then signed and processed for release.

93. The Android documentation (from <http://developer.android.com/guide/developing/building/index.html>) for building and running Android applications describes the general process for a typical build as follows:

- “The Android Asset Packaging Tool (aapt) takes your application resource files, such as the AndroidManifest.xml file and the XML files for your Activities, and compiles them. An R.java is also produced so you can reference your resources from your Java code.”
- “The aidl tool converts any .aidl interfaces that you have into Java interfaces.”
- “All of your Java code, including the R.java and .aidl files, are compiled by the Java compiler and .class files are output.”
- “The dex tool converts the .class files to Dalvik byte code. Any 3rd party libraries and .class files that you have included in your project are also converted into .dex files so that they can be packaged into the final .apk file.”
- “All non-compiled resources (such as images), compiled resources, and the .dex files are sent to the apkbuilder tool to be packaged into an .apk file.”
- “Once the .apk is built, it must be signed with either a debug or release key before it can be installed to a device.”
- “Finally, if the application is being signed in release mode, you must align the .apk with the zipalign tool. Aligning the final .apk decreases memory usage when the application is running on a device.”

94. The basic correspondence between the Java application development and execution platform components and the Android application development and execution platform components is shown in the figure below. This figure shows that in both cases, Java source code is compiled to Java bytecode. In Android, there is an additional step in which Java

bytecode is converted to .dex bytecode. This same conversion is applied to Java class library bytecode. Then, in both scenarios, bytecode is executed on the end-user platform by a virtual machine operating under control of an operating system installed on a physical computing device.



I. Android's Adoption of the Java Platform

95. The Java programming language, the Java SDK, and Java libraries are all used in Android. The way each of these is used by Android and the importance of the inventions protected by intellectual property asserted in this lawsuit is summarized below.

96. ***Java programming language.*** As discussed above, the Java programming language is one of two leading general purpose programming languages. The other is C++, including the C subset which could also be considered a third programming language that shares core programming, compilation, and execution characteristics with C++. Of these, Java is a

clear choice for Android because it supports managed execution (execution in an established virtual machine) and the “write once, run anywhere” paradigm that allows developers to use any of several platforms to produce software that runs on any of a range of platforms. Because Android, *unlike iPhone and other competitors*, is intended to run on diverse platforms produced by independent manufacturers, support for “write once, run anywhere” is central to the entire Android strategy. (I will address Windows Mobile separately below.)

97. Another highly important advantage of Java is the well-established and highly-trained developer community. The popularity and widespread use of Java in education and in industry provide a large community of experienced Java programmers. By tapping into this abundant pool of experts, Google’s Android has been able to attract a large number of application developers. Because Java is standardized and associated with a reliable SDK, experienced Java developers can readily build Android applications using skills they developed through Java programming for other platforms. These factors are undoubtedly critical to Google’s success in filling the Android Market with a wide range of applications, which is in turn a significant part of Google’s success with Android overall.

98. ***Java development platform.*** Google adopted existing and proven development tools that were known to be attractive to developers and, because they were already built, tested, and widely accepted, improved Android’s speed to market. Eclipse, discussed below, and the Java compiler, are complex software systems that required substantial teams to develop, support, and improve over time.

99. As I described earlier, the Java Development Kit (JDK) provides the backbone of the Android SDK—the system requirements for installing the Android SDK include a Java Integrated Development Environment (IDE), and appropriate versions of the JDK. Google also recommends and takes advantage of Eclipse, a popular IDE developed by IBM for Java. To leverage Eclipse for Android developers, Google provides a specific Android Development Tools (ADT) plug-in. (See <http://developer.android.com/sdk/requirements.html>.)

100. **Java class libraries and APIs.** The Java class libraries are distributed with the JDK and form an integral part of the Java development environment. The class libraries and their APIs are the result of extensive effort to build software components that are easy to use, general in purpose, and effective for solving a range of common software development. The Java Community Process (JCP) website (<http://www.jcp.org>) lists many projects illustrating the effort involved in defining good APIs. For example, one of two goals of JSR 166 Concurrency Utilities is “To standardize a simple, extensible framework that organizes commonly used utilities into a small enough package to be readily learnable by users and maintainable by developers.” (See <http://jcp.org/en/jsr/detail?id=166>.)

101. Google copied substantial portions of Java APIs, as described in more detail in this report. This made Android programming more familiar to Java programmers, furthering interest in Android application development and speeding the development of new applications. In addition, Google did not have to invest the time and effort required to formulate good APIs or incur the associated costs and time-to-market delays. (See, e.g., GOOGLE-01-00019511 at 511 (“[J]ava provides a nice safety net and faster app development and debuggability. (this is based on experience developing hiptop – java saved us a pretty crazy amount of time).”).)

J. Google’s Decision to Adopt Java for Android and Why Alternatives Are Suboptimal

102. The Google Android team made a decision to use a specific programming language and associated platform for Android development and Android applications. From a technical standpoint, key Android project requirements that are relevant to the choice of language and platform are:

- Ability to run compiled code on heterogeneous devices produced by multiple vendors. (See, e.g., GOOGLE-04-00042610 at 611 (“Java solves a lot of the portability issues C++ has.”); 7/7/2011 Swetland Dep. 33:19-20 (“I think the VM as a platform feature in Android was most important from a portability standpoint”).)
- Require widespread acceptance among platform vendors who need to produce compatible device drivers and custom branding. (See, e.g., GOOGLE-12-00003871 at 873 (“Carriers require Java in their terminal.”).)

- Trained and capable developer community with millions of developers. (*See, e.g.*, GOOGLE-01-00025376 at 419 (“Strategy: Leverage Java for its existing base of developers.”); GOOGLE-02-00111218 at 218 (“Java has very little fragmentation, and it’s adoptable. If we play our cards right, we can also leverage not only existing developers, but applications as well.”).)
- Tools and development platform that is familiar to developers and runs on accepted developer systems (e.g., leading operating systems). (*See, e.g.*, GOOGLE-01-00019511 at 512 (“Java is more accessible than C++ There is more standardization in tools and libraries.”); GOOGLE-01-00019527 at 527 (Android is building a Java OS. We are making Java central to our solution because a) Java, as a programming language, has some advantages because it’s the #1 choice for mobile development b) there exists documentation and tools c) carriers require managed code d) Java has a suitable security framework.”).)
- Short time to market and low risks of selling a buggy consumer electronics device. (*See, e.g.*, GOOGLE-01-00075935 at 935 (“We will ship a more stable product sooner if we do as much as possible in Java”).)

103. These goals stem from the strategic decision Google made not to contract with a single device manufacturer and the importance of Android applications in attracting consumers to Android. (*See* GOOGLE-12-00079180 at 185 (“We are forming an alliance with interested parties to make this free platform the de facto standard for modern handsets.”).) Without an application marketplace that was competitive with Apple iPhone, Android would not compete favorably with iPhone overall. (*See* GOOGLE-00302808 at 808 (“We will increase partner revenue share . . . but we need to provide at least 70% to the developer to make [Android] Market competitive with other app stores (eg. iPhone App Store).”).) Further, as iPhone had been gaining heavily against BlackBerry (as documented elsewhere in this report), any delays in reaching the market would put Android at a further disadvantage to iPhone. (*See* GOOGLE-00383073 at 075 (analyzing how many BlackBerry users would likely switch to Android as opposed to iPhone for their next smartphone purchase).) In particular, in addition to needing to make up additional market share, an unchecked increase in the iPhone market share would obviously decrease the financial motivation of potential Android developers.

104. There are also some additional, possibly secondary, self-evident criteria. For example, it would be an advantage to Google Android to select a programming language for Android that appealed to developers and that might generate media buzz in the engineering

community. Further, an ability to engage the scientific and engineering research community through a platform that was freely available to them would clearly benefit Google.

105. Among the Google Android needs, the importance of time-to-market should not be underestimated. (*See, e.g.*, GOOGLE-01-00019529 at 530 (“It is widely believed that if an open platform is not introduced in the next few years then Microsoft will own the programmable handset platform.”); GOOGLE-29-00002338 (“Risk that people may flock to other platforms if we wait too long.”); 7/27/2011 Rubin Dep. 179:14, (“I was under incredible schedule pressure”); 7/27/2011 Rubin Dep. 180:1-12 (“[Y]ou have a window of opportunity in smartphones You have to ship as soon as feasibly possible. I mean, you go to extraordinary lengths to ship sooner, because it’s a very dynamic market. And it could shift directions at any time So my job as . . . the architect of this business concept was to just do everything that I possibly could to get my solution to the market in the shortest time possible.”).) Based on time to market considerations, Google needed to adopt an existing programming language and platform.

106. Google considered several alternative programming languages and accompanying platforms. (*See, e.g.*, 4/5/2011 Rubin Dep. 22:23-23:5 (“We could use the C programming language. We could use the C++ programming language. We could use JavaScript at the time, we had some exploratory discussions around JavaScript. Also Microsoft C Sharp as the programming language. We also discussed briefly Lua as a programming language and Python.”).) Google apparently considered all in the following list:

- **JavaScript** (*See, e.g.*, 7/7/2011 Swetland Dep. 44:6-7 (“We had been playing with JavaScript” as an alternative to the Java programming language.); 5/16/2011 Bornstein Dep. 50:3-4 (“I think the other main candidates were JavaScript and C++.”).)
- **Lua**
- **Python** (*See, e.g.*, 7/7/2011 Swetland Dep. 44:7-8 (“Python might have been discussed at one point” as an alternative to the Java programming language.).)
- **C#** (*See, e.g.*, GOOGLE-01-00019527 at 528 (“Abandon our work and adopt MSFT CLR VM and C+ language.”); 4/5/2011 Rubin Dep. 107:18-19 (“[W]e explored briefly some options with Microsoft and C Sharp.”).)

- **C/C++** (*See, e.g.*, 7/7/2011 Swetland Dep. 44:1-3 (“I thought C++ could be very useful and also the core of the OS . . . at the user space level was written in C++ at the time.”); 5/16/2011 Bornstein Dep. 50:3-4 (“I think the other main candidates were JavaScript and C++.”).)
- **Objective C**, often abbreviated to **Obj-C**
- **“The iPhone model,”** which uses **Obj-C**
- **Java** (*See, e.g.*, GOOGLE-04-00055169 at 170 (“we are building a java based system: that decision is final”).)

107. While C/C++ does not occur in the primary documents I studied to determine which alternatives Google considered, I have included it because it is an extremely popular programming language. While some or all of the alternatives considered may also lead to alternative Android platforms that infringe one or more of the asserted patents, Google recognized that there are also technical reasons why none of the alternative languages considered were satisfactory when Google chose to go mainstream with Android’s adoption of Java.

108. **JavaScript.** JavaScript is a scripting language for Web applications that was originally designed by Brendan Eich, then at Netscape (see, e.g., www.mozilla.org/js/). Although the word “Java” occurs as part of the name “JavaScript,” there is no other fundamental connection between the languages. As far as I know, the name “JavaScript” was part of a marketing agreement only.

109. Continuing evolution of the JavaScript language is managed by ECMA Standards Committee TC39 (of which I am a member). Modern web applications use JavaScript for client-side functionality and all modern web browsers contain JavaScript interpreters. Because JavaScript interpreters require source code, web applications that use JavaScript transfer JavaScript source code from the web server to the browser.

110. JavaScript execution is also relatively inefficient, in comparison to Java or compiled languages. This is not necessarily the result of lack of implementation effort – JavaScript has dynamic capabilities that are normally associated with slow execution speed, including runtime object construction, variable parameter lists, function variables, dynamic script creation (via eval), object introspection (via for ... in), and source code recovery. (My Stanford

colleague Dan Boneh has developed a cryptographic library for JavaScript that I believe runs slowly in comparison with Java, in spite of his efforts and those of his graduate students.) Mozilla, for example, hosts two JavaScript implementations, Spidermonkey and Rhino. Even if each of these uses an intermediate format to speed the implementation, I know of no standard intermediate format that is used to exchange a form of compiled or partially compiled JavaScript. Among other reasons, the inefficiency of JavaScript execution and the fact that applications would need to be provided in source code seem to disqualify JavaScript as the basic application development language for a smartphone project such as Android.

111. It is interesting to note that a substantial portion of Google JavaScript programming actually relies on Java, Java libraries, and the JDK. Google produces and uses Google Web Toolkit (GWT), “a development toolkit for building and optimizing complex browser-based applications.” (See <http://code.google.com/webtoolkit/>.) As the web site further explains, “GWT is used by many products at Google, including Google AdWords and Orkut.” The basic operation of GWT is that “It lets you write client-side applications in Java and deploy them as JavaScript.” (See <http://code.google.com/webtoolkit/overview.html>.) As further noted on the same page, “The GWT SDK contains the Java API libraries, compiler, and development server.” (*Id.*)

112. **Lua.** The Lua programming language (<http://www.lua.org/>) is “a scripting language born in 1993 at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro in Brazil.” (See R. Ierusalimschy, L. H. de Figueiredo, W. Celes, The evolution of Lua, Proceedings of ACM HOPL III (2007) 2-1–2-26, *available at* <http://portal.acm.org/citation.cfm?id=1238846>.) It is a relatively untested programming language, in comparison with the others considered for Google Android.

113. Lua appears as the 50th most popular language in the Tioble list for 2006 and 10th for 2011. Still, it is not widely taught in universities and my estimate is that even if it is tenth in popularity according to the Tiobe measure based on web sites, there is a large difference

in the number of experienced programmers for the top 3-5 languages in this list and the languages that are tenth or lower on this list.

114. In summarizing properties of the language, the main proponents of Lua wrote, “Semantically, Lua has many similarities with Scheme, even though these similarities are not immediately clear because the two languages are syntactically very different.”

(R. Ierusalimsky, L. H. de Figueiredo, W. Celes, The evolution of Lua, Proceedings of ACM HOPL III (2007) 2-1–2-26, *available at* <http://portal.acm.org/citation.cfm?id=1238846>.) In this regard, Lua is related to JavaScript, which is essentially based on Scheme, a functional programming language, and Self, an object-oriented language. Two important features that are common to all three languages are anonymous functions and full lexical scoping. In addition, like Scheme and JavaScript, “Lua is dynamically typed: variables do not have types; only values have types.” (R. Ierusalimsky, L. H. de Figueiredo, W. Celes, The evolution of Lua, Proceedings of ACM HOPL III (2007) 2-1–2-26, *available at* <http://portal.acm.org/citation.cfm?id=1238846>.) In addition, the main data-structuring mechanism of Lua, called tables, are associative arrays that are analogous to JavaScript objects.

115. The implementation of Lua uses a one-pass compiler to produce bytecode for a register-based virtual machine. This virtual machine is implemented in ANSI C and tuned for portability rather than performance: “To be portable across many different C compilers and platforms, Lua cannot use several tricks commonly used by interpreters, such as direct threaded code. Instead, it uses a standard while-switch dispatch loop. Also, at places the C code seems unduly complicated, but the complication is there to ensure portability. The portability of Lua's implementation has increased steadily throughout the years, as Lua got compiled under many different C compilers in many different platforms (including several 64-bit platforms and some 16-bit platforms).” (R. Ierusalimsky, L. H. de Figueiredo, W. Celes, The implementation of Lua 5.0, Journal of Universal Computer Science 11 #7 (2005) 1159–1176, *available at* http://www.jucs.org/jucs_11_7/the_implementation_of_lua/jucs_11_7_1159_1176_defigueiredo.pdf.)

116. From a software development standpoint, one interesting characteristic of Lua is a mechanism of called fallbacks. This feature allows programmers to extend the semantics of the language in some unconventional ways. For example, fallbacks allow the user to add different kinds of inheritance to the language. Typically, unusual language features like this have advantages for some kinds of programming situations and drawbacks for others. For example, a feature that allows programmers to add different kinds of inheritance to the language could be useful with those new kinds of inheritance are needed, but problematic in other situations because they could make it harder for one programmer to read another programmer's code, as is often needed in software development, testing, and maintenance. Whatever the advantages or disadvantages of this and other unusual features of Lua, using a language that has not been widely adopted represents a significant risk.

117. Overall, Lua would have been an unusual and risky choice for Google Android because it is not as widely used as other languages, it has the same drawbacks as JavaScript and Python, and it does not have anywhere near the number of experience developers as the other languages considered.

118. **Python.** Python (<http://www.python.org/>) is another programming language that is sometimes referred to as a scripting language. Python is considered an interpreted language (e.g., <http://docs.python.org/tutorial/index.html>), although there appear to be implementations that translate Python to the Java and .NET virtual machines. (See <http://www.python.org/>.) As a flexible, dynamic language with generally slow performance, Python has largely the same drawbacks as an Android implementation or application development and deployment language as JavaScript. If an implementation of Python based on the JVM were used, this would infringe the asserted patents as they are used in the Oracle JVM, or the performance degradation associated with non-infringing workarounds work be significant, as measured by the performance measurements discussed elsewhere in this report.

119. **C#.** Microsoft's C# (see <http://msdn.microsoft.com/en-us/vcsharp/aa336809>) and other .Net languages are analogous to Java in many of the respects relevant to the Android

platform and the Google Android team decision to use Java. According to the Microsoft web site (<http://www.microsoft.com/net/overview.aspx>), the .Net Framework consists of:

- **Common Language Runtime** – ...an abstraction layer over the operating system
- **Base Class Libraries** – ... for common low-level programming tasks
- **Development frameworks and technologies** – reusable, customizable solutions for larger programming tasks.

120. The .Net framework supports C#, a language that has some similarities to both C++ and Java, and a number of other programming languages that are all compiled to produce intermediate code that is executed under control of a virtual machine. Specifically,

“In the .NET environment, there are two distinct parts to compilation. The first part entails a programmer compiling and optimizing with the language compiler (C#, Visual Basic®, or Visual C++) to generate MSIL. The second part involves the MSIL being fed to the just-in-time (JIT) compiler or NGEN, which reads the MSIL and then generates optimized native machine code.”

(See <http://msdn.microsoft.com/en-us/magazine/cc163855.aspx>.)

121. In other words, the C# and .Net approach is similar in that a compiler produces an executable intermediate format analogous to Java bytecode. This bytecode, in a format analogous to class files, is supplied to a virtual machine. While there are technical differences, the advantages of C# or other .Net languages are similar to Java. However, C# and .Net are proprietary products of Microsoft Corporation and Google Android would have had to negotiate terms with Microsoft. In addition, the C# developer community is not as extensive as Java, the development environment is not available under the same terms, and so on. Therefore, while there are technical reasons that C# and other .Net languages might be reasonable alternatives, there are business issues that would have to be addressed. Because Windows Mobile has been less successful than Android, it seems reasonable to expect that Android would not have been as successful if Microsoft languages and platforms had been used.

122. **C/C++.** The C programming language was originally designed and implemented from 1969 to 1973, as part of the Unix operating system project at Bell Laboratories. C was

designed by Dennis Ritchie, one of the original designers of Unix, the popular and successful operating system leading to Linux. C++ is an object-oriented extension of the C language that was originally designed by Bjarne Stroustrup in the early 1980s. (Stroustrup was then working at Bell Laboratories. I also worked at Bell Laboratories in the summer of 1983, when the language was called “C with Classes,” and then 1984-1988.) The compiler and execution models for these languages are similar. Because C is a subset of C++, a C++ compiler may also serve as a C compiler.

123. The C and C++ languages are compiled to native code. If separate program units are compiled independently, then the output files containing native code are linked by a linker to produce an executable file (such as would have a “.exe” extension under Windows). This paradigm is not satisfactory for Android because Android applications must run on heterogeneous hardware platforms developed independently by independent smartphone vendors. One consequence of this is that developers must be able to build applications that compile and execute correctly on multiple platforms. This suggests a need for developers to have several emulators, one for each target platform. This complicates development. In addition, if a new vendor wishes to sell Android phones, under this set of conditions, developers would have to adapt their applications to make sure they work properly on the new hardware platform.

124. There are also problems with providing applications to users in this scenario. If smartphone applications are developed in C/C++, application developers could distribute source code or compiled code. However, it is undesirable to distribute source code for several reasons. For example, source code generally reveals the structure of an application and does not protect the intellectual property of developers.

125. For all of these reasons, a language that is compiled to native code does not provide “write once, run anywhere”—it does not fit the Android model because it does not support direct development on multiple platforms and it does not provide the ability to run compiled code on heterogeneous devices produced by multiple vendors.

126. **Objective C and “The iPhone model.”** Objective-C is an extension of the C programming language that provides “syntax for defining classes, and methods, as well as other constructs that promote dynamic extension of classes.” (See http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/.)

127. Objective-C is used at Apple and it is the programming language for iPhone applications. Specifically, programmers interested in developing iPhone apps are instructed to learn to program using Objective-C and are supplied with a specific development environment. (See, e.g., “Your First iOS Application” at <http://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhone101/iPhone101.pdf>.) Specifically, “The tool you use to create applications for iOS is Xcode—Apple’s IDE (integrated development environment).” (*Id.* at 11.)

128. Objective-C is compiled and executed using essentially the same process as C and C++. In fact, the popular gcc C compiler can compile Objective-C source code, when the appropriate command-line option is supplied. An example showing Objective-C source code, a command line for compiling source code, and the command-line input for executing the resulting native executable is presented in a chapter of Stephen G. Kochan’s book, Programming in Objective-C 2.0, Rough Cuts, 2nd Edition, available at <http://www.informit.com/articles/article.aspx?p=1271260>. An Objective-C runtime environment that provides compiled code with a number of operations that can be invoked at run time is documented on Apple’s website (e.g., <http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/>).

129. Because Objective-C is compiled to native code, and it is an alternative superset of C, it suffers from the same drawbacks described above for C and C++. A language that is compiled to native code, Objective-C does not provide “write once, run anywhere”—it does not fit the Android model because it does not provide the ability to run compiled code on heterogeneous devices produced by multiple vendors. The reason that Objective-C is suitable for

iPhone and iPad applications is that Apple provides a single hardware platform that runs appropriately compiled native code. iPhone applications do not need to run on heterogeneous devices produced by multiple vendors.

130. **Java.** As Google recognized, Java provides a perfect solution for Android. (*See, e.g.,* GOOGLE-01-00019511 at 511-13 (listing “[r]easons to shift to a primarily Java API”); GOOGLE-04-00055169 at 169-72 (listing advantages of Java in Android “manifesto”).) Java provides the ability to run compiled code on heterogeneous devices produced by multiple vendors because Java source code is compiled to bytecode that runs on any platform with an appropriate virtual machine. In addition, Java virtual machines have been developed for standard architectures and hardware instruction sets, making Java an attractive option for device vendors who might be concerned about producing and marketing devices with unproven software.

131. Java has a trained and capable developer community with millions of developers. Java has been the leading programming language in colleges and universities. Many companies, large and small, use Java, as illustrated by the Tiobe data, for example. Java developers building Android applications can use familiar developer tools, such as Eclipse, run on accepted developer platforms such as Linux, Windows, and MacOS computers. (*See* GOOGLE-00302662 at 662 (Android features “[r]ich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE.”).)

132. Java allowed Google Android to reduce time to market because the Java language and platform was already in widespread use. Java development tools were available and familiar to developers. Because Java code analysis and Q/A procedures were well known to experienced architects, designers, programmers, and project managers, Java minimized the risk of Google Android bringing a buggy consumer electronics device to market, when compared with unproven software alternatives. Google Android team members clearly agreed with the conclusion that Java is the best programming language and platform when they selected Java. Further, the

success of Android over competing smartphones using different software platforms suggests that selecting Java was a successful strategic decision.

133. *Unsuitability of Non-infringing Java alternatives.* While the advantages of the Java language and platform are clear, Google Android could hypothetically have used the Java programming language and the Java compiler, but developed its own theoretically non-infringing execution platform. I have analyzed the design alternatives for achieving this goal, as described below, and my opinion is that this was not a feasible or successful option.

134. Once Java is chosen, Google Android could hypothetically choose to either (i) supply some form of executable intermediate form object code to a virtual machine on the device, or (ii) compile Java bytecode to native code for each device. Within the second alternative, this compilation could be done in two ways: (ii-a) compile Java bytecode or some translation of bytecode to native code on the device, or (ii-b) compile Java bytecode or some translation of bytecode to native code off the device, prior to transfer onto the device.

135. Hypothetical alternative (ii-b), which produces native executable code for each application before it is installed on the mobile device, suffers from the same drawbacks as the compiled languages C/C++ and Objective-C discussed above. For the reasons discussed above in connection with those languages, this does not appear to be a feasible alternative because for example, it fails to achieve “write once, run anywhere.” Moreover, to the extent that this involves developing new compilers and potentially new compilation technology for Java bytecode, there are additional time-to-market and software fragility risks that may make this an even less attractive alternative than adopting C/C++ or other C-based languages.

136. Hypothetical alternative (ii-a), which involves compiling Java bytecode or some translation of bytecode to native code on the device, would have to be developed carefully to avoid infringing the asserted patents. Further, if one considers that additional Java patents were owned by Sun Microsystems (now Oracle), there are additional constraints that limit the set of alternatives available to Google in designing Android.

137. In the final hypothetical alternative, Google Android could attempt to produce a non-infringing Java-based alternative by individually working around each of the claims of each of the asserted patents. But as noted above, additional Java patents held by Sun Microsystems (now Oracle) add constraints that limit the set of alternatives available to Google in designing Android.

138. I will further address non-infringing alternatives in my patent expert report.

V. THE COPYRIGHTS IN SUIT AND DETAILED OPINIONS

A. Statement of Opinions and Findings

139. In preparing this section of the report, I reviewed Application Program Interface (“API”) specifications, source code, and documentation relating to various versions of the Java Platform Standard Edition and Android. I also reviewed a number of the pleadings, documents produced during discovery, and deposition transcripts from this case.

140. Based on my understanding of copyright law, it is my opinion that multiple elements of the Java API specifications contain copyrightable expression. These elements include the selection, coordination, and arrangement of the Java packages; the class names and definitions, fields, methods, and method signatures contained in each package; and the English prose text that explains each of the above elements. Similarly, the source and object code that implements the Java core libraries contains copyrightable expression.

141. It is my further opinion that significant portions of the Android API specifications bear substantial similarity to the Java API specifications. Similarly, the Android source code that implements the Android specifications contains these same elements. Other Android source code is substantially similar to Oracle’s copyrighted source code or to decompiled Oracle object code. Thus I conclude, based on my understanding of copyright law, that this element of copyright infringement has been met both as to Google and as to third parties who reproduce or distribute Android materials that they receive from Google.

B. Principles of Law

142. I have been informed that under the law, a copyright owner has the right to exclude others from reproducing, preparing a derivative work from, distributing, performing, or displaying, the copyrighted work. I understand that the term derivative work refers to a work based on one or more preexisting works, in which the preexisting work is recast, transformed, or adapted.

143. It is my understanding that to establish direct copyright infringement, a plaintiff must prove that the plaintiff is the owner of the copyright and that the defendant copied original elements of the copyrighted work.

144. I understand that to prove that the defendant copied the plaintiff's work, the plaintiff may show that the defendant had access to the plaintiff's copyrighted work and that there are substantial similarities between the defendant's work and the plaintiff's work. Where a high degree of access is shown, a lower standard of proof of substantial similarity is required.

145. I further understand that in assessing similarity, courts consider both quantitative similarity—how much was copied—as well as qualitative similarity—the significance of what was copied. I understand that copying is considered “de minimis” and is not actionable if the copying is neither quantitatively nor qualitatively significant.

146. I have further been informed that copyright protects original expression. Facts, ideas, procedures, processes, systems, methods of operation, concepts, principles, and discoveries cannot be copyrighted. A compilation of individually unprotectable elements, however, may be copyrightable. A compilation is a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole constitutes an original work of authorship.

147. I understand that Oracle is not asserting copyright infringement of the Java language as such but of creative works that are written in and use the Java programming language.

148. I have been informed that when an idea and its expression are indistinguishable, or “merged,” the expression will only be protected against nearly identical copying. Likewise, if technical constraints dictate particular expression, then that expression is only protected against virtually identical copying.

149. I have assumed that while Google's explanation for its implementation of Java APIs is its desire to provide a familiar, “compatible” programming environment for Java programmers, that commercial objective does not excuse otherwise actionable misappropriation.

150. I have been informed that a defendant is liable for vicarious copyright infringement if the defendant has profited directly from the infringing activity and has the right and ability to supervise the infringing activity, whether or not the defendant knew of the infringement.

151. I have been informed that a defendant is liable for contributory copyright infringement if the defendant knows or should have known of infringing activity by another and induces, causes, or materially contributes to the activity.

152. I understand that an infringement is considered willful when the plaintiff has proved that the defendant engaged in acts that infringed the copyright, and the defendant knew that those acts infringed the copyright.

C. Ownership

153. I understand that Oracle owns the copyrights in the code, documentation, specifications, libraries, and other materials that comprise the Java platform. Oracle has obtained copyright registrations for a number of Java-related works, including multiple editions of the book *The Java™ Language Specification*; the 1996 book *The Java™ Application Programming Interface, Volume 1*; and numerous versions of the Java Development Kit (“JDK”) software, each of which contains thousands of computer program and documentation files. I understand that each version of the JDK builds upon previous versions and that Oracle and its predecessors separately registered copyrights for each major JDK release. The Java-related code, documentation, specifications, libraries, and other materials that I reviewed in preparing this report bear Sun or Oracle copyright notices, making it apparent that Oracle owns the copyrights to these materials.

154. I understand that third parties contributed to portions of the Java platform and that these third parties have either assigned or licensed the copyrights in their contributions to Oracle. I further understand that Oracle is not asserting copyright infringement of certain third party materials in the Java platform. This report only covers materials in which Oracle has alleged infringement.

D. Access

155. I understand that Google had access to Oracle's copyrighted works in a variety of sources. The Application Programming Interface specifications and binary installation packages for Oracle's JDKs have been available on the websites of Oracle and its predecessors for many years. (See <http://download.oracle.com/javase/1.5.0/docs/api/>; <http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html>.)

Oracle has also made much of the JDK source code available on its website for non-commercial use under research licensing programs. (See *id.*) All of the JDK source code that I have personally observed has contained a copyright notice for Sun Microsystems or Oracle. Therefore anyone who sees the code is aware that Oracle has legal rights associated with it.

156. I understand that Google made it a design goal for Android to mimic the structure of Oracle's Java libraries. According to the Android Compatibility Definition, "Android follows the package and class namespace conventions defined by the Java programming language." (See GOOGLE-00296163.) In order for developers to build Android applications, Google requires them to download the JDK. (See <http://developer.android.com/sdk/requirements.html>, listing JDK 5 or JDK 6 as a requirement for the Android Software Development Kit.)

BEGIN GOOGLE ATTORNEYS' EYES ONLY

157. I further understand that in 2007, Google hired third party contractors at Noser Engineering to implement core library code compatible with a subset of the Java Standard Edition, version 5 libraries. (See GOOGLE-00392204-12, a statement of work between Google and Noser dated March 28, 2007.) I understand from documentation that Google engineers supervised Noser's work and worked closely with Noser engineers on the effort. (See, e.g., GOOGLE-00392213, a statement of work between Google and Noser dated January 29, 2008 that requires two Noser engineers to work at Google's Mountain View office.) In order to determine whether Noser had delivered what Google had contracted for, both Google and Noser engineers would have to have access to the Java SE API specifications.

158. I understand that according to Google's records, a number of Google employees and contractors who worked on Android previously had access to Sun's Java code. Google intended that developers contribute Android code to the Apache Harmony project. (*See* GOOGLE-00296500-03 (list of Android developers who contributed to the Apache Harmony project).) I understand that to avoid allegations that contributors copied Sun or Oracle code, the Harmony project requires contributors to complete questionnaires indicating their past exposure to Java libraries. Google produced copies of such questionnaires for several Android developers. (*See, e.g.*, GOOGLE-00320083-88 (Authorized Contributor Questionnaire for Carlo U. Nicola); GOOGLE-00320116-21 (Authorized Contributor Questionnaire for Markus Pilz); GOOGLE-00320162-66 (Authorized Contributor Questionnaire for Florian Brunner).)

159. I also understand that Joshua Bloch, who was an architect of the Java platform at Sun, now works for Google. (7/8/2011 Bloch Dep. 7:16-17.) Not only did Bloch, as an employee, have access to Sun Microsystems code generally, but his name appears in the source code of several Java library files, including `java.util.arrays.java`, as an author.

160. I further understand that when asked whether Google consulted any J2SE 1.5 documentation to evaluate the work it got from Noser, Android programmer Dan Bornstein responded: "we had the printed materials, we had some amount of Javadoc." (5/16/2011 Bornstein Dep. 161:1-2.) He further testified that he knew that the Javadoc¹ was copyrighted. (*Id.* at 161:11.)

END GOOGLE ATTORNEYS' EYES ONLY

E. Refresher on Object-Oriented Programming and the Java Language

161. As an "object-oriented" platform, Java's features are divided into "classes" of software "objects." Each class is created in human-readable source code before being compiled into machine-readable byte code.

¹ Javadoc is a tool that extracts developers' comments from java code for use as documentation. As Bornstein used the term, Javadoc refers to the Java API specification documentation generated using the Javadoc tool.

162. Within the Java platform, classes are grouped into “packages” that provide related features. `Java.io`, for instance, is a package that contains the classes and Interfaces that implement input and output functions. `Java.nio` is a separate package that handles a “new” set of input and output functions. `Java.nio.channels` is a sub-package of `java.nio`.

163. The classes are organized hierarchically within Java such that a subclass may inherit characteristics from its parent. For example, the class `java.io.InputStreamReader` is a subclass of `java.io.Reader`.

164. Many classes contain “fields” for storing data. For example, the class `java.io.FilterOutputStream` contains a field called “out,” which stores, as the name might suggest, a data output stream to be filtered.

165. Each class also contains “methods” associated with particular tasks. For example, the class `java.lang.String` contains a method defined by the Java API specification as follows: “`public boolean contentEquals(CharSequence cs)`.” This method, named “`contentEquals`,” compares a string of text to another sequence of characters and returns a value of “TRUE” if the character sequences are the same and “FALSE” if not. Each method will typically take variables called “parameters” as input. In the above example, `contentEquals` accepts a `CharSequence` object named “`cs`.” The combination of a method’s name along with the number and types of its parameters is known as the method’s “signature.” “Public” refers to the visibility of the method to other objects. “Boolean” is the “return type” of the method; in this case, it means that the method returns “TRUE” or “FALSE.”

166. As an additional example, the class `java.lang.Runtime`² contains a method defined with the line “`public Process exec(String command, String[] envp, File dir)` throws `IOException`.” The “`exec`” method above takes three parameters as input instead of just one. The method performs the function of executing the specified string “`command`” in a separate process with the specified environment variables “`envp`” and the specified working directory

² See Exhibit Copyright-F for documentation on `java.lang.Runtime`.

“dir.” In this example, the return type of the method is a “Process” object, and the method generates an “IOException” in case of an error.

167. Each method is implemented in source code based on one or more algorithms. An algorithm is a step-by-step procedure for accomplishing a goal. While it is my understanding that algorithms are not protected by copyright, the creative expression in the source code that carries out an algorithm is protected.

168. Java also includes a reference type known as an Interface that contains only constants and abstract (*i.e.*, declared, but not yet implemented) methods. Classes can implement Interfaces by providing code that carries out, or implements, their abstract methods. For clarity I will capitalize the Interface type to distinguish it from the general use of the term “interface.” One Interface can extend another, which means that Interfaces, like classes, may be arranged hierarchically. `Java.util.Collection`, described in more detail below, is an example of an Interface for managing groups of objects.

169. I understand that code remains protected by copyright when it is compiled into machine-executable object code, which is also known as executable or binary code, and that this is true both for native object code that is directly executed on hardware and for object code designed for execution in a virtual machine.

F. The Value of Application Program Interfaces

170. The Java core libraries include classes of code that perform common functions such as input-output, networking, and handling collections of data. By harnessing Java core libraries, programmers do not have to rewrite code that was already written to perform existing functions.

171. An Application Program Interface for a class library might be compared to a guidebook that explains how to use the library. It describes each class and its specified behaviors. It includes the name of each class in the library. It also defines each class’s relationship—typically hierarchical—to other classes and to packages of classes. For each class in the library, an API describes the fields and methods that are exposed to other classes. The API

indicates the defined input parameters and output “return” type, if any, for each publicly exposed method. Although it is not essential for the operation of the code, each parameter is generally given a descriptive name to convey its intended use.

172. An effective API architect, like any writer, must keep his or her audience in mind. The audience includes application developers and system designers who utilize APIs as they create their systems. An API may also serve as the design of a library, to be used by developers who write source code that implements the library. If an API is written clearly and efficiently, it will be easier to learn and use. Developers will be more likely to adopt the underlying platform. The more developers use a platform, the more applications they will write, and the more applications that exist, the greater the appeal of a platform for end users.

173. The strength of the Java API and its associated tools is likely a key reason for the platform’s popularity.

BEGIN GOOGLE ATTORNEYS’ EYES ONLY

174. Google acknowledged as much in a 2006 presentation: “6M Java developers worldwide. Tools and documentation exist to support app development without the need to create a large developer services organization. There exist many legacy Java applications. The wireless industry has adopted Java, and the carriers require its support. Strategy: Leverage Java for its existing base of developers.” (GOOGLE-01-00025576 at 584.)

END GOOGLE ATTORNEYS’ EYES ONLY

175. An API for a library has many expressive aspects apart from the library’s functionality. The expressive elements of an API include package, class, and method names; the selection, naming, and order of parameters; and documentation that explains how the API works. An API specification does not actually run on a computer; rather, it describes a set of rules that the code implementing the library must follow. It serves as a point of contact between the library code and the developers who will call on it.

176. Software companies expend considerable time and resources to craft elegant APIs to address their problems. Joshua Bloch, a former Sun engineer who now works for Google,

wrote in a 2005 presentation that an API can be “among a company’s greatest assets.” (OAGOOGL0100219512.) “Customers invest heavily: buying, writing, learning.” (*Id.*)

G. Copyrightable Expression in the Java Platform

BEGIN GOOGLE CONFIDENTIAL

177. Given the value of APIs, it is no surprise that Bloch and other API architects have recognized the importance of copyright protection. When asked about the significance of copyright protection for the specifications he wrote at Sun, Bloch replied: “[I]f someone else were to take this prose and publish it for profit, Sun would probably be upset, and with good reason.” (7/8/2011 Bloch Dep. 62:23-25.)

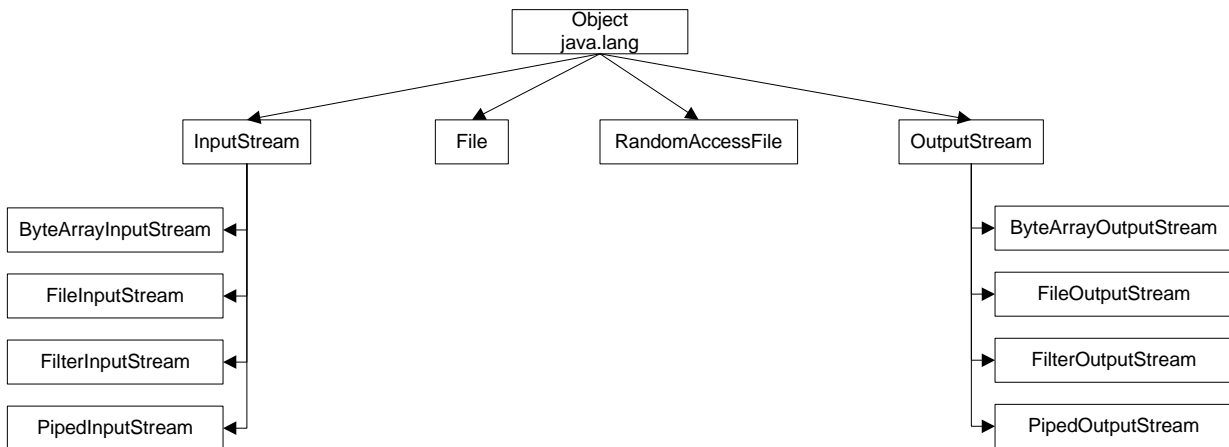
END GOOGLE CONFIDENTIAL

178. The Java API Specifications contain considerable original expression in the selection, coordination, and arrangement of the Java packages, as well as the class names, definitions, fields, methods, and method signatures contained in each package. The English prose text that explains each of the above elements represents yet another layer of expressive content.

179. When the architects of the Java API decided to include a particular set of classes in the platform and to give these classes particular names, they chose from a wide range of expressive options. For example, the package `java.io` is concerned with the idea of handling input and output. The Java core libraries express that idea by including the classes `java.io.PipedInputStream` and `java.io.PipedOutputStream` to handle input and output. The architects of the Java API did not have to include these classes as part of the standard Java platform; it was a choice. Core libraries for other platforms such as C++ do not include such classes.

180. The decision, moreover, to organize these classes into the designers’ chosen hierarchy reflects the original choices of the designers. As an illustration, a subset of the class hierarchy for `java.io` is diagrammed below. Note that the division between input and output streams occurs at a very high level on the tree. The designers wanted to convey the idea that

“input” classes, which are close together on the tree, share common characteristics, just as, for example, mammals have more in common with other vertebrates than they do with invertebrates.



181. Indeed, subclasses of `InputStream` have an input method that the designers called “read” while subclasses of `OutputStream` have an output method that they called “write.” The designers could have placed all “ByteArray-,” “File-,” “Filter-,” and “Piped-” streams together on the tree before dividing between Input and Output. Such an arrangement would not have affected the functionality of the classes, but it would not have been an equivalently elegant solution. By branching initially on the dimension of input versus output, yet maintaining symmetry within the hierarchy, the designers created a library that is functional and an API that is aesthetically pleasing and easy to understand.

182. The creation of the `java.io` hierarchy presented additional opportunities for creative expression, as the designers could have used other names, such as “get” and “put” instead of “read” and “write” for input and output operations. Note also that `InputStream`, `File`, `RandomAccessFile`, and `OutputStream` could have been defined as Interfaces rather than classes.

183. The decision to arrange Java classes into particular packages reflects yet another conscious choice of expression. Extending the example above, rather than handling data input and output in the package `java.io`, the designers could have separated input and output classes into different packages. The classes’ visibility to one another may have then changed, but the

substantive functionality of these classes would not. What does *not* go into a package may be just as important as what does. When the designers wanted to add new input and output classes to the Java API, they created a new package, `java.nio`, rather than modifying `java.io`. Given that `java.io` is already relatively large—particularly at the levels of the hierarchy not diagrammed above—additional classes may have overwhelmed developers.

184. Within each class, there are even more opportunities for original expression. The choice of which fields and methods to include in a class, as well as what to name them, provides many options for a developer. It is true that developers tend to use somewhat descriptive names for fields and methods so that readers can follow their code. As Joshua Bloch has indicated, “names should be largely self-explanatory.” (OAGOOGL0100219527.) If names are not only descriptive in themselves, but follow a consistent pattern established by the API author, developers will readily understand their meaning. These names are not dictated, however, by the function that the underlying elements perform. A computer can execute a method called “`ct`” just as easily as it can execute a method called `compareTo`. The reason for choosing descriptive method names is so that a human audience can learn them more easily.

185. The components of a method, which include the method signature, embody additional creativity. Design choices abound in the selection and arrangement of input parameters for a method. In the method `java.lang.Runtime.exec(String command, String[] envp, File dir)` mentioned above, for example, the designers could have used different input parameters, and they could have organized them in a different order. Indeed, the Java API specification includes multiple versions of the “`exec`” method that differ only in their parameter lists. The computer treats each version differently on this basis alone. The versions include:

- `exec(String command)`
- `exec(String[] cmdarray)`
- `exec(String[] cmdarray, String[] envp)`
- `exec(String[] cmdarray, String[] envp, File dir)`
- `exec(String command, String[] envp)`

- `exec(String command, String[] envp, File dir)`

186. The decision to include these six particular versions of “exec” when a smaller number may have sufficed embodies considerable creativity. If “exec” had many more parameters, on the other hand, the number of available versions would rapidly become unwieldy. Just as a convoluted description in a textbook might confuse students, a long list of similar parameters will surely confuse developers.³ To avoid confusion, Bloch recommends that designers use consistent parameter ordering across methods (OAGOOGL0100219544) as illustrated in the example above. Line (d) above could have been specified as `exec(String[] envp, String[] cmdarray, File dir)`, but developers used to the order of line (c) would have likely overlooked the change and made mistakes in their programs.

187. The names of parameters are even more expressive than the names of classes and methods because developers do not have to use parameter names to make use of the API. For example, a developer who wants to make use of the method `java.lang.String.charAt(int index)`⁴ would not actually use the name “index” in a program. Rather, the developer would use an integer, or a variable of integer type. The developer might use `charAt(5)` or, if “x” were an integer, `charAt(x)`. Parameter names are chosen not to make a method work, but to help explain how a method works.

188. Not to be overlooked, the source code that implements the Java API presents tremendous opportunity for creative expression. As Bloch stated in his presentation, “code should read like prose.” (OAGOOGL0100219527.) Like prose writers, developers typically can choose many ways to achieve a desired result.

BEGIN GOOGLE CONFIDENTIAL

189. In the words of Joshua Bloch, the “role of documentation in an API is to communicate precisely what that API does and how to accomplish that function.” (7/8/2011 Bloch Dep. 85:13-15.)

³ Bloch’s presentation acknowledges as much in suggesting that “three or fewer parameters is ideal.” (OAGOOGL0100219545.)

⁴ This method returns the character in a particular position, or index, within a string of text.

END GOOGLE CONFIDENTIAL

190. The comments that describe the code in the Java platform present yet another instance of the Java API designers' creative expression. Comments are not "functional" in the sense that they are not compiled for execution by the computer; rather they are written purely for the use of a human audience. If a Java source code file is formatted properly, a tool called JavaDoc can extract the developers' comments for use as documentation for the code in question and the API that the code implements.

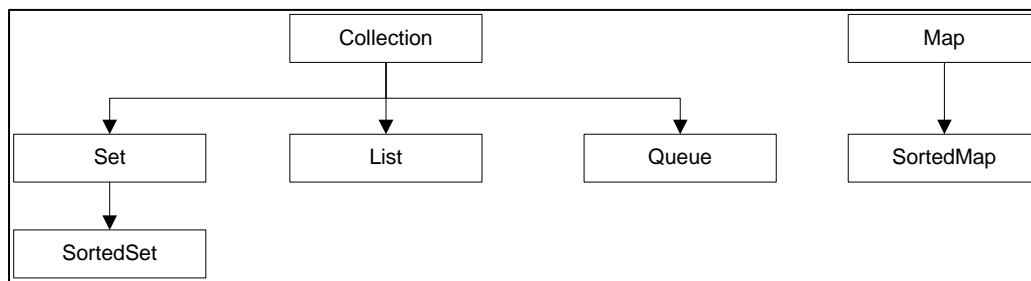
191. While there were previous libraries for other languages that provide related functionality, the Java API represents an original work. An attempt to port the library class-by-class from another language, would not have been feasible. The difficulty in porting an API—even one with similar functionality—from one platform to another exemplifies the need for careful planning and expression in API design. As Bloch points out, "transliterating" an API between programming languages is rarely effective. (*See* OAGOOGL0100219532.)

BEGIN GOOGLE CONFIDENTIAL

192. "[I]f you have an API that was written in one program language, say C++, and you are trying to provide the same functionality in a second programming language, say Java, there is a natural tendency among programmers to simply take every class in the C++ program and every method in each of those classes and provide corresponding classes and methods in Java. And if you do this, you will usually get gobbledygook." (7/8/2011 Bloch Dep. 87:13-21.)

END GOOGLE CONFIDENTIAL

193. For example, the Java Collections Framework addresses a task that is common across many programming languages: storing and manipulating groups of data as a single unit. The Java API handles collections using a set of abstract Interfaces that form a hierarchy. The first few levels of the hierarchy from the Java Standard Edition 5.0 Development Kit are diagrammed below.



Subset of Java Collections Framework

194. The Collection interface is the root of the collections hierarchy.⁵ This Interface defines basic operations shared by all collections, including methods to return the size of a collection, to determine whether it is empty, and to determine whether it contains a particular element or group of elements. Many other operations, such as adding an element to a collection, are optional.

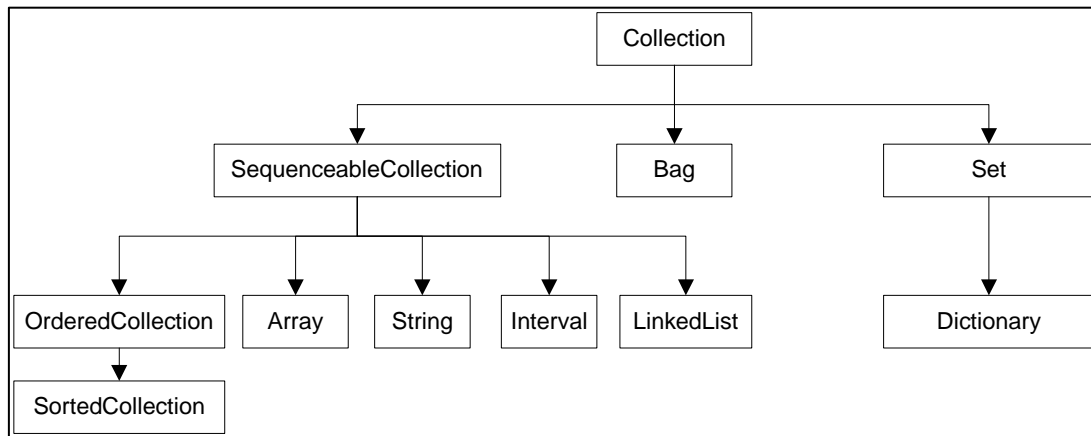
195. In Java, a Set is a Collection that cannot contain duplicate elements. A List is an ordered Collection that may or may not contain duplicates. A Queue is a Collection used to hold multiple elements prior to processing that typically places its elements in FIFO (first-in, first-out) order. A Map is an object that associates unique keys with particular values the way a dictionary associates words with their meanings. SortedSet and SortedMap are sorted versions of their parent Interfaces.

196. Other platforms handle collections differently. For example, subsets of the Smalltalk collections hierarchy⁶ and the C++ Standard Template Library⁷ are diagrammed below.

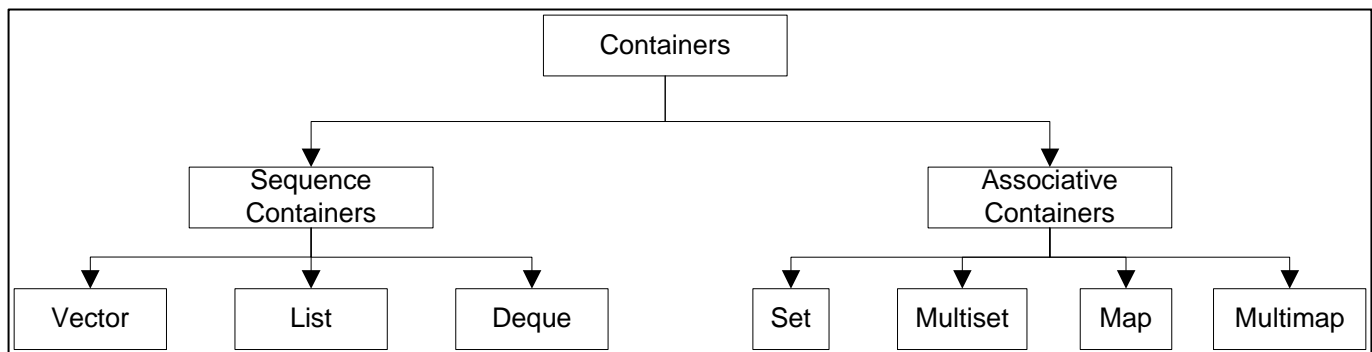
⁵ The descriptions of the Java Collections interfaces are adapted from <http://download.oracle.com/javase/tutorial/collections/interfaces/index.html>.

⁶ For descriptions of the SmallTalk Collections hierarchy, see William R. Cook, *Interfaces and Specifications for the Smalltalk-80 Collection Classes*, OOPSLA 1992.

⁷ See generally, Silicon Graphics International, Standard Template Library Programmer's Guide, available at http://www.sgi.com/tech/stl/table_of_contents.html.



Subset of Smalltalk Collections Hierarchy



Subset of C++ Standard Template Library

197. At a glance it is clear that the three APIs follow different organizational schemes. For example, SmallTalk's Dictionary, which contains key-value pairs much like Java's Map, is a subclass of SmallTalk's Set. By contrast, C++'s Map occupies the same level as Set, and Java's Map is not even a true Collection.

198. Different types of collections are present in each API. OrderedCollection is a SmallTalk analog of Java's List. SmallTalk's SortedCollection, however, has no sorted List counterpart in Java. Nor do SmallTalk's Bag or C++'s multiset—essentially unordered collections that allow duplicates—have an equivalent in the JDK 5 API. These collections libraries take different creative approaches to describing their solutions to a common problem.

199. It is notable that the Java API did not originally include an extensive collections framework. (See <http://download.oracle.com/javase/tutorial/collections/intro/index.html>.) The initial Java Development Kit included only Vector, array, and Hashtable classes. When the Collections Framework was added for Java Development Kit 1.2, these classes were retrofitted to implement the new interfaces. For example, Vector was retrofitted to implement List. The fact that the Java API has evolved further supports the conclusion that there were many expressive options available to the architects when it was created.

H. The Android APIs Compared to the Java APIs

200. For this comparison, I examined the API specifications of JDK 5, which contains 166 packages, and Android version 2.2 ("Froyo" or API Level 8), which contains 157 packages. I used JDK 5 as the comparand because I understand that, when Google created Android, it intended Android's core libraries to implement a subset of the Java 2, Standard Edition, version 5 APIs. Of course, JDK 5 is based on earlier versions of the Java platform, which, I understand, Sun also copyrighted. I understand that for subsequent versions of Android, Google updated its packages to cover Java 6 APIs. (See <http://source.android.com/source/initializing.html>, indicating developers will need "JDK 6 if you wish to build Gingerbread or newer; JDK 5 for Froyo or older.")

BEGIN GOOGLE ATTORNEYS' EYES ONLY

(See also GOOGLE-00381507 at 508, a status report on Android core libraries.)

END GOOGLE ATTORNEYS' EYES ONLY

My same analysis would apply for JDK 6.

201. My visual comparison of Android's API package specifications (available at <http://developer.android.com/reference/packages.html>) with Oracle's copyrighted Java API package specifications (for example, available at <http://download.oracle.com/javase/1.5.0/docs/api/>) demonstrates that Java and Android organize their classes and packages in an identical, hierarchical fashion. (See Exhibit Copyright-A.) An alphabetical list of packages is included in the upper-left hand frame, and a list of associated classes is included in the frame below. As noted *infra*, many of the included package names are identical. The central frame in both specifications describes a particular package and explains each of its constituent elements. The word "package" precedes the package name, and below the name is a description of the package contents. Below the package description in both specifications is a table listing each interface, class, and exception for a package, with a short explanation of each.

202. The following Android package specifications are substantially similar to Oracle's copyrighted Java API package specifications⁸:

⁸ I understand that in this case, Oracle has chosen not to assert copyright infringement of several other Java SE packages, in some cases because Oracle uses these packages under license from third parties or allows third parties to utilize these packages under permissive terms. These packages include: java.math, java.util.concurrent, java.util.concurrent.atomic, java.util.concurrent.locks, javax.xml, javax.xml.datatype, javax.xml.namespace, javax.xml.parsers, javax.xml.transform, javax.xml.transform.dom, javax.xml.transform.sax, javax.xml.transform.stream, javax.xml.validation, and javax.xml.xpath. These packages are not included in my analysis.

java.awt.font	java.nio.channels.spi	java.util	javax.net.ssl
java.beans	java.nio.charset	java.util.jar	javax.security.auth
java.io	java.nio.charset.spi	java.util.logging	javax.security.auth.callback
java.lang	java.security	java.util.prefs	javax.security.auth.login
java.lang.annotation	java.security.acl	java.util.regex	javax.security.auth.x500
java.lang.ref	java.security.cert	java.util.zip	javax.security.cert
java.lang.reflect	java.security.interfaces	javax.crypto	javax.sql
java.net	java.security.spec	javax.crypto.interfaces	
java.nio	java.sql	javax.crypto.spec	
java.nio.channels	java.text	javax.net	

203. The 37 packages listed above have identical names and structures and contain substantially similar characteristics in Java and Android. The similar elements include Oracle's class names, definitions, fields, and methods.

204. For example, in both Java and in Android, the package java.nio contains ten classes (Buffer, ByteBuffer, ByteOrder, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, MappedByteBuffer, and ShortBuffer) and four exceptions (BufferOverflowException, BufferUnderflowException, InvalidMarkException, ReadOnlyBufferException), all with identical names. For additional examples, see Exhibits Copyright-B-D (packages java.lang, java.io, java.security).

205. Drilling down further into this example, the class java.nio.IntBuffer has nearly identical method signatures in both Java and in Android as shown below:

Methods in java.nio.IntBuffer (Java version)	Methods in java.nio.IntBuffer (Android version)
static IntBuffer allocate(int capacity)	static IntBuffer allocate(int capacity)
int[] array()	final int[] array()
int arrayOffset()	final int arrayOffset()
abstract IntBuffer asReadOnlyBuffer()	abstract IntBuffer asReadOnlyBuffer()
abstract IntBuffer compact()	abstract IntBuffer compact()
int compareTo(IntBuffer that)	int compareTo(IntBuffer otherBuffer)
abstract IntBuffer duplicate()	abstract IntBuffer duplicate()
boolean equals(Object ob)	boolean equals(Object other)
abstract int get()	abstract int get()
abstract int get(int index)	abstract int get(int index)
IntBuffer get(int[] dst)	IntBuffer get(int[] dst)
IntBuffer get(int[] dst, int offset, int length)	IntBuffer get(int[] dst, int dstOffset, int intCount)
boolean hasArray()	final boolean hasArray()
int hashCode()	int hashCode()
abstract boolean isDirect()	abstract boolean isDirect()
abstract ByteOrder order()	abstract ByteOrder order()

Methods in java.nio.IntBuffer (Java version)	Methods in java.nio.IntBuffer (Android version)
abstract IntBuffer put(int i)	abstract IntBuffer put(int i)
IntBuffer put(int[] src)	final IntBuffer put(int[] src)
IntBuffer put(int[] src, int offset, int length)	IntBuffer put(int[] src, int srcOffset, int intCount)
IntBuffer put(IntBuffer src)	IntBuffer put(IntBuffer src)
abstract IntBuffer put(int index, int i)	abstract IntBuffer put(int index, int i)
abstract IntBuffer slice()	abstract IntBuffer slice()
String toString()	(Inherited from parent class)
static IntBuffer wrap(int[] array)	static IntBuffer wrap(int[] array)
static IntBuffer wrap(int[] array, int offset, int length)	static IntBuffer wrap(int[] array, int start, int intCount)

206. Of the 25 methods in the Java version of java.nio.IntBuffer, 24 are present in the Android version. Fifteen rows in the above table are *identical* with respect to method names and parameter lists. Four of the Android methods add the keyword “final” to prevent subclasses from modifying the method’s behavior, but they are otherwise identical. Five other Android methods make minor changes to the names of parameters, but not their types or order. Changing variable names without changing program structure is a trick that students might use in an attempt to hide copying in programming assignments. In this case, there is not even an attempt to hide the copying; it is literal and readily apparent.

207. In addition to the method signatures shown above, the explanatory comments for each method are also substantially similar in Java and Android. See Exhibits Copyright-E-F (covering java.security.KeyPair & java.lang.Runtime respectively) for examples of this phenomenon. As an illustration, the Java API specification describes the method java.security.KeyPair.getPrivate as follows: “Returns a reference to the private key component of this key pair.” The Android spec uses the following prose: “Returns the private key.” Other examples of similar comments can be found throughout the API specifications.

208. Based on my understanding of copyright law, I conclude that the Android API specification literally copies a significant amount of protectable expression from the API specification in the Java Standard Edition 5.0 Development Kit.

I. Android Source Code Compared to the Java APIs

209. Android’s API specifications contain class names, definitions, fields, methods, and explanatory text that are substantially similar to the corresponding elements in Oracle’s Java

API specifications, leading to an inference of copying. The Android code that implements the Android specifications includes these copied elements.

210. Google source code and documentation can be found in Google's repository at <http://android.git.kernel.org/>. For example, the source code that implements the java.security package in Android 2.2 can be found in the Android repository at `dalvik/libcore/security/src/main/java/java/security`⁹. This branch of the repository contains source code files, written in the Java programming language, with which Google intends to implement to the java.security classes, interfaces, and exceptions defined in the Java API specification.

211. As a further example, according to the Java API Specification¹⁰, the class `java.security.ProtectionDomain` has two constructors¹¹ with the following signatures:

- `ProtectionDomain(CodeSource codesource, PermissionCollection permissions)`
- `ProtectionDomain(CodeSource codesource, PermissionCollection permissions, ClassLoader classloader, Principal[] principals)`

212. Furthermore, the specification requires six methods for this class with the following signatures:

- `public final CodeSource getCodeSource()`
- `public final CodeSource getCodeSource()`
- `public final Principal[] getPrincipals()`
- `public final PermissionCollection getPermissions()`
- `public final PermissionCollection getPermissions()`
- `public final PermissionCollection getPermissions()`

⁹ Citations to the Android source repository are shortened and mirror the file paths shown in <http://android.git.kernel.org>. For example, “dalvik/libcore/security/src/main/java/java/security” maps to <http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=libcore/security/src/main/java/java/security/ProtectionDomain.java;h=1e85c4a86d2aba29ffe841b88d2fb5dba7b0e579;hb=refs/heads/froyo>.

¹⁰ See <http://download.oracle.com/javase/1.5.0/docs/api/index.html?java/security/ProtectionDomain.html>.

¹¹ A constructor is a pseudo-method that creates an object.

213. As shown in Exhibit Copyright-G, the Android file `dalvik/libcore/security/src/main/java/java/security/ProtectionDomain.java` copies the above lines from the Java API Specification nearly verbatim and then includes code that purports to implement the behavior required by this specification. The copied elements include the selection, name, parameter types, and order of the constructors as well as the selection, name, and order of the methods.

214. `ProtectionDomain.java` is one example of how Android code copies from the Java APIs. In addition, there are hundreds of source code files that implement the Java packages in a similar manner. Additional examples can be found in the Android repository at the following locations.

215. For Android 2.2 (“Froyo”):

- `dalvik/libcore/security/src/main/java/java/security;`
- `dalvik/libcore/security/src/main/java/javax/security/cert;`
- `dalvik/libcore/security/src/main/java/org/apache/harmony/security;`
- `dalvik/libcore/math/src/main/java/java/math;`
- `dalvik/libcore/math/src/main/java/org/apache/harmony/math;`
- `dalvik/libcore/luni/src/main/java/java;`
- `dalvik/libcore/luni/src/main/java/org/apache/harmony/luni;`
- `dalvik/libcore/luni-kernel/src/main/java/java/lang;`
- `dalvik/libcore/luni-kernel/src/main/java/org/apache/harmony/kernel;`
- `dalvik/libcore/luni-kernel/src/main/java/org/apache/harmony/lang;`
- `dalvik/libcore/nio/src/main/java/java.`

216. For Android 2.3 (“Gingerbread”):

- `libcore/luni/src/main/java/java/security;`
- `libcore/luni/src/main/java/javax/security/cert;`
- `libcore/luni/src/main/java/org/apache/harmony/security;`
- `libcore/luni/src/main/java/java/math;`

- libcore/luni/src/main/java/java;
- libcore/luni/src/main/java/org/apache/harmony/luni;
- libcore/luni/src/main/java/java/lang;
- libcore/luni/src/main/java/org/apache/harmony/kernel;
- libcore/luni/src/main/java/org/apache/harmony/lang;
- libcore/luni/src/main/java/java/nio.

217. Notably, the content of each of these branches mirrors the content of each corresponding Java package.

218. Google has created and distributed via its online repository works written in native code, in addition to Java code, that show substantial similarity to Oracle's copyrighted works. For example, Google makes and distributes dalvik/vm/native/java_lang_Class.c, which is based on Oracle's java.lang.Class specification. Other examples include:

- dalvik/vm/native/java_lang_Object.c
- dalvik/vm/native/java_lang_reflect_AccessibleObject.c;
- dalvik/vm/native/java_lang_reflect_Array.c;
- dalvik/vm/native/java_lang_reflect_Constructor.c;
- dalvik/vm/native/java_lang_reflect_Field.c;
- dalvik/vm/native/java_lang_reflect_Method.c;
- dalvik/vm/native/java_lang_reflect_Proxy.c;
- dalvik/vm/native/java_lang_Runtime.c;
- dalvik/vm/native/java_lang_String.c;
- dalvik/vm/native/java_lang_System.c;
- dalvik/vm/native/java_lang_Throwable.c;
- dalvik/vm/native/java_lang_VMClassLoader.c;
- dalvik/vm/native/java_lang_VMThread.c; and
- dalvik/vm/native/java_security_AccessController.c.

219. These source files implement portions of the Java APIs in the C programming language instead of the Java programming language. They are complementary to the Java-language implementations. Because each of the above source code files copies substantial protectable expression from Oracle's API specifications, it is my opinion that they are derivative works.

220. Many of the above files can also be found, compiled into executable code, on Android-compatible devices. For example, Samsung makes a portion of the source code for its SGH-I897 "Captivate" handset available at <https://opensource.samsung.com/>. A comparison reveals that the source code file ProtectionDomain.java taken from the file SGH-I897_OpenSource.tar.gz on the above site is textually identical to the corresponding file in Google's GIT repository. The same holds true for at least the files:

- dalvik/libcore/security/src/main/java/java/security/Certificate.java
- dalvik/libcore/security/src/main/java/java/security/CodeSource.java
- dalvik/libcore/security/src/main/java/java/security/Key.java
- dalvik/libcore/security/src/main/java/java/security/Permission.java
- dalvik/libcore/security/src/main/java/java/security/Policy.java
- dalvik/libcore/security/src/main/java/java/security/ProtectionDomain.java
- dalvik/libcore/security/src/main/java/org/apache/harmony/security/PolicyEntry.java
- dalvik/libcore/security-kernel/src/main/java/java/security/AccessController.java
- dalvik/libcore/security-kernel/src/main/java/java/security/AccessControlContext.java

221. Documentation contained in many of the source code packages on Samsung's site indicates that Google provides the majority of the source code for Samsung Android devices. The "readme.txt" file contained in SGH-I897_OpenSource.tar.gz instructs developers who want to build code for their devices to first download a complete source code set from Google's source.android.com website and then overwrite specific Google modules with the code from

Samsung's site. The developer is then instructed to "make," or compile, the code. The contents of "readme.txt" is reprinted in Exhibit Copyright-I.

222. Other device makers provide a subset of the code for their devices and similarly refer developers to Google for the bulk of the code for use in builds. (*See, e.g.* "README" from US670(Thunder)_Android_Froyo_USA_USC_Opensource.zip, retrieved from <http://www.lg.com/global/support/opensource/opensource.jsp>.)

223. As noted above, it is even possible to find the names of copyrighted Java classes on shipping Android devices. For example, Samsung Nexus S includes the java.security.ProtectionDomain class within /system/framework/core.odex on the device. Other devices have it as well. This indicates that code derivative of the Java core libraries appears on Android devices.

224. The Android SDK, distributed by Google, contains one or more complete system images that contain compiled versions of the Android code that includes the elements copied from the Java APIs.

225. I further understand that Google exercises considerable influence over the code that appears on devices. Google's Android Compatibility Definition ("CD") requires that all devices deemed Android-compatible comply with the CD and pass Google's Compatibility Test Suite ("CTS"). (*See* GOOGLE-00296158.) I understand that Google forbids device manufacturers from using the trademark "Android" in association with their devices unless they pass the CTS. Furthermore, the CD strongly discourages device makers from modifying Google's Android platform code before copying it onto their devices. "Device implementers are strongly encouraged to base their implementations on the 'upstream' source code available from the Android Open Source Project. While some components can hypothetically be replaced with alternate implementations this practice is strongly discouraged, as passing the CTS tests will become substantially more difficult." (*Id.*) Thus I conclude that Android devices available in the marketplace are more likely than not to contain code from Google that manufacturers utilize with few, if any, modifications.

226. Google's warning against changing code, coupled with the device makers' instructions that users download the bulk of their code from Google, makes it quite likely that the majority of the Android platform code that Google provides actually ends up compiled on end user devices.

227. I understand that Google participates in the design and build of some device makers' handsets, and provides the final Android build to the OEM. (*See, e.g.*, GOOGLE-22-00281510-45.)

228. It is my understanding that if a manufacturer, developer, or end user copies infringing materials to a device, then that person is a direct infringer.

229. I understand that after it provides its platform code for free to device makers, Google profits from advertising and value-added services that it sells on top of the Android platform.

BEGIN GOOGLE ATTORNEYS' EYES ONLY

(*See, e.g.*, Patrick Brady, "Android Strategy and Partnerships Overview," June 2009, GOOGLE-22-00171914 at 924 ("Android isn't a new product to monetize; it's a new medium to drive monetization on existing products.").)

END GOOGLE ATTORNEYS' EYES ONLY

230. In summary, it is my understanding that vicarious liability requires third party copying, profit to the defendant, and an ability for the defendant to supervise the infringing activity. I conclude that Google uses its definition of "Android-compatible" to control the code that appears on devices and profits directly from device makers' use of materials that copy original expression from Oracle's Java API specifications.

231. It is my further understanding that contributory infringement requires third party copying, knowledge by the defendant, and material contribution or inducement. I conclude that Google strongly encourages device makers and developers to use code that Google itself supplies for reproduction and distribution on their devices. This code, moreover, copies original expression from Oracle's Java API specifications.

J. Android Source Code Compared to Java Source Code

232. Google has distributed by way of Android and Android-related websites source and object code substantially similar to Oracle's source code or to decompiled Oracle object code, as described below.

233. Two Android source code files contain lines that are identical to those in Oracle's java.util.arrays.java. These files are: /dalvik/libcore/luni/src/main/java/java/util/TimSort.java and /dalvik/libcore/luni/src/main/java/java/util/ComparableTimSort.java. Specifically, the method "rangeCheck" is identical both the Oracle and Android files.

rangeCheck() from Oracle java.util.arrays.java lines 1318-326 [spacing adjusted for comparison]	rangeCheck() from Android TimSort.java lines 915-923 [spacing adjusted for comparison]
<pre>private static void rangeCheck(int arrayLen, int fromIndex, int toIndex) { if (fromIndex > toIndex) throw new IllegalArgumentException("fromIndex(" + fromIndex + ") > toIndex(" + toIndex + ")"); if (fromIndex < 0) throw new ArrayIndexOutOfBoundsException(fromIndex); if (toIndex > arrayLen) throw new ArrayIndexOutOfBoundsException(toIndex); }</pre>	<pre>private static void rangeCheck(int arrayLen, int fromIndex, int toIndex) { if (fromIndex > toIndex) throw new IllegalArgumentException("fromIndex(" + fromIndex + ") > toIndex(" + toIndex + ")"); if (fromIndex < 0) throw new ArrayIndexOutOfBoundsException(fromIndex); if (toIndex > arrayLen) throw new ArrayIndexOutOfBoundsException(toIndex); }</pre>

234. The method's name, parameter list, internal variables, and logic are identical in Java and in Android.

235. It is true that rangeCheck is only nine lines long (excepting comments, which are also substantially similar between the files) as originally printed, and Oracle's arrays.java comprises 3,180 lines of code. Nevertheless, rangeCheck is qualitatively significant to arrays.java, as it is called nine times by other methods in the class.

236. The similarity between the two rangeCheck methods is notable for a number of reasons. First, the order of the logic is identical. The method performs three separate checks against the bounds of an array using an "if... then" pattern. Thus there are six possible orders in which these checks could have occurred in the program logic. The order is identical in Java and Android. Second, the names of the parameters and internal variables are the same in both Java

and in Android. Third, the Java and Android versions have identical, inconsistent spacing. There are spaces on either side of every “+” character, except the last, which has no spaces. I note that `rangeCheck` was not included in Oracle’s public API specification for `java.util.arrays`. (See <http://download.oracle.com/javase/1.5.0/docs/api/java/util/Arrays.html>.) This is expected because `rangeCheck` is declared to be a “private” method.

237. I understand that the author of `arrays.java`, Josh Bloch, left Sun Microsystems and now works for Google. Not only did Bloch, as an employee, have access to Sun Microsystems code generally, but his name is listed in the source code of `java.util.arrays.java` as an author.

BEGIN GOOGLE CONFIDENTIAL

238. During a deposition, Bloch admitted to writing `TimSort` while at Google. (See 7/8/2011 Bloch Dep. 162:8-163:7.) When asked whether he accessed Sun code while working on `TimSort`, Bloch responded: “I don’t have a recollection, but I’m perfectly willing to believe that I did. You know, I think the similarity of the signature, the fact that, you know, the three arguments are in the same order and have the same name, you know, is a strong indication that it is likely that I did.” (7/8/2011 Bloch Dep. 181:9-14.) It is notable that by comparing the method signature and the names and the order of the parameters, Bloch concluded that copying had likely occurred even though he could not recall whether he had performed the copying.

END GOOGLE CONFIDENTIAL

239. The fact that the specification for `rangeCheck` does not appear to have been public, combined with the fact that the method’s author left Sun to work for Google, provides a strong indication that, at the very least, the Android designers had access to Oracle’s copyrighted works, brought copies with them, and derived code from them.

240. Much like the `java.security` source code described above, `TimSort.java` and `ComparableTimSort.java` appear in the source code archives for the Samsung SGH-I897 “Captivate” and other handsets. For the same reasons discussed above, it thus seems quite likely that these files, as distributed by Google, end up compiled on Android devices.

241. In addition to the rangeCheck method discussed above, the source code for eight Android programs appears to be nearly identical to decompiled Oracle class files.

242. A decompiler's function is opposite that of a compiler. A compiler converts human-readable source code into machine-executable object code. Source code files written in the Java programming language (".java" files) are compiled into binary files ending with the suffix ".class". A decompiler, on the other hand, generates source code from object code. Compilation removes all explanatory prose from source code, as a computer cannot execute comments. Thus, when an object code file is decompiled, the comments that were present in the original source code are missing.

243. While decompilers for some programming languages generate source code that bears little resemblance to the original source, Java decompilers can generate source code fairly similar to the original input. This allows potential copiers who have access to a program's object code, but not its source code, to generate copies similar to the original program source. In this case, the object code for the Java class libraries is available for download from Oracle's website.

244. Software consultants from Johnson-Laird Inc. investigated whether there were any files in Android that were obtained by decompiling Oracle object code. The consultants used a decompiler called JAD on a binary version of the Java class libraries from JDK 5. The consultants then used automated file comparison tools to search for similarities between the decompiled Oracle class files (now ending with suffix ".jad") and the Android source code. I have reviewed their report, which describes their methodology and the results of their analysis. The analysis indicated eight Android program files, listed in the table below, that bear significant textual similarity to the corresponding decompiled Oracle class files.

Decompiled Oracle Class File	Corresponding Android File
/sun/security/provider/certpath/PolicyNodeImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harmony/security/tests/support/cert/PolicyNodeImpl.java
/sun/security/acl/AclEntryImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harmony/security/tests/support/acl/AclEntryImpl.java
/sun/security/acl/AclImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harmony/security/tests/support/acl/AclImpl.java
/sun/security/acl/GroupImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harmony/security/tests/support/acl/GroupImpl.java

Decompiled Oracle Class File	Corresponding Android File
/sun/security/acl/OwnerImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harm ony/security/tests/support/acl/OwnerImpl.java
/sun/security/acl/PermissionImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harm ony/security/tests/support/acl/PermissionImpl.java
/sun/security/acl/PrincipalImpl.class	/dalvik/libcore/support/src/test/java/org/apache/harm ony/security/tests/support/acl/PrincipalImpl.java
/sun/security/acl/AclEnumerator.class	/dalvik/libcore/support/src/test/java/org/apache/harm ony/security/tests/support/acl/AclEnumerator.java

245. My own visual comparison of these files, detailed in Exhibits Copyright-J-Q, shows that the similarity is striking. Indeed, once one adjusts for spacing, the files are nearly identical on a line-by-line basis.

246. For example, the decompiled version of PolicyNodeImpl.class (PolicyNodeImpl.jad) is 247 lines long, and the Android version of PolicyNodeImpl.java is 257 lines long. (See Exhibit Copyright-J.) Within these lines, the order of the methods is identical, as is the method logic. Variable names are also largely identical, right down to the rather unusual use of generic names such as “Set set1” and “boolean flag1.” A professional developer might choose longer, more descriptive names. A decompiler, on the other hand, might create such variable names as a rule. Thus the variable names constitute strong evidence of copying.

247. Excluding the boilerplate text at the beginning of each file, the biggest differences between the two files are that (1) they are part of different packages and (2) the Java version uses four “import” statements while the Android version uses one generic “import” statement. The other differences are largely cosmetic: one file might omit a set of optional braces, while the other might squeeze two lines of otherwise identical code into one.

248. It would be extraordinarily unlikely to implement a program this long that exhibits this degree of similarity to another program without copying. The same generally holds true for the rest of the files listed in the table above, documented at Exhibits Copyright-J-Q.

249. Two Android files contain comments that are nearly identical to comments in corresponding Java files, as shown in the table below and at Exhibits Copyright R-S. As noted above, comments, unlike other portions of code, are not executed by the computer. Because they are not “functional,” comments represent one of the most expressive elements of a computer

program. The decision to include particular comments is generally an intentional one rather than a product of external constraint.

Oracle Java File	Corresponding Android File
/java/security/CodeSource.java	/dalvik/libcore/security/src/test/java/org/apache/harmony/security/tests/java/security/CodeSourceTest.java
/java/security/cert/CollectionCertStoreParameters.java	/dalvik/libcore/security/src/test/java/tests/security/cert/CollectionCertStoreParametersTest.java

250. As shown in Exhibits Copyright-R and S, Android does not place the entire block of comments from a file in one area; rather, it spreads them across multiple lines of code. This makes copying more difficult to detect. Within each area of similarity, however, the only difference in the left and right columns in the Exhibit is tags that provide HTML formatting in the Java version.

251. The developers of Android files CodeSourceTest.java and CollectionCertStoreParametersTest.java apparently extracted the comments either from Oracle's code or from Oracle's API specifications for these classes. The comments appear to be used in explaining how Android's testing code tests the functionality of Android's implementation of the java.security API.

252. I conclude, based on my understanding of copyright law, that Google has distributed programs that copy original expression from Oracle's copyrighted Java source code and thus derive from Oracle's work. Moreover, I conclude that, to the extent that this code ends up on Android devices, Google encourages device makers, developers, and end users to further reproduce and distribute Oracle's copyrighted materials. Because it would have been effectively impossible for someone to see Oracle's code without seeing the copyright notice on it, it is reasonable to conclude that any copying on Google's part was willful.

VI. COPYRIGHTED FEATURES FORM BASIS FOR CUSTOMER DEMAND FOR ANDROID

253. Sun Microsystems's (now Oracle's) copyrighted features, including the Java library API and portions of the source code implementing the library (discussed in my report), form a basis for consumer demand. In particular, customer demand for Android devices is driven by the availability of a wide range of appealing applications ("apps") to consumers, which is what Android Market is today. (Of course, Google developed its own set of Android applications such as Gmail and Google Maps. (*See, e.g.,* 7/12/2011 Morrill Dep. 38:9-39:7.) In order to provide these applications to Android device users, Google must attract developers to build such applications for Android in the first place.

254. The Java API attracts developers because it is well known and familiar to them. By incorporating the Java API, Google Android leverages the Java API to draw on the extensive community of experienced Java application developers and attract developers to the Android platform. Google also used Java to attract carriers and OEMS, as described below.

255. In adopting Java, Google took steps to maintain and provide developers with the core Java library that form the basis for Oracle's copyright infringement claim against Google as detailed above.

BEGIN GOOGLE ATTORNEYS' EYES ONLY

A. Google Understood that Applications Would Drive Consumer Demand for Android and Java Would Play a Central Role

256. The Android documents I have studied show that Google understood that applications would drive consumer demand for Android.

257. As discussed above in the report section describing the success of Android, Android is distinguishable from competing mobile platforms (other than iPhone) by the Android application developer community, the Android Market, and the large number of varied applications available to Android users.

258. Google recognized the importance of applications and made design decisions, such as adoption of the Java platform, accordingly.

259. For example, an August 16, 2006 E-mail from key Android developer Brian Swetland (Google) (GOOGLE-04-00055098) reads as follows (emphasis added):

“- platform: priority one is user experience
 - if we do not ship a compelling experience
 (dialer, pim, maps, whatever) we FAIL
 - the platform must serve the apps & experience
 - writing great apps must be simple
 - “it is complicated because it is powerful” is a lousy answer to
 “why is it so hard to do X”.”

The Java platform and the class library API specifications make “writing great apps” “simple” because they provide application developers a familiar development environment that speeds and eases application development by providing pre-written code that application developers can just call upon rather than programming from scratch.

260. Another August 5, 2005 E-mail from Brian Swetland (Google) (GOOGLE-12-00000537 at 539) similarly highlights the importance of the Java platform in attracting “some really good java application development and user experiences”:

“The JVM is going to be a central piece of the system we’re building, not some little add-on on the side – so we can provide some really good java application development and user experiences.”

The “JVM” is a reference to the Java virtual machine. To be clear, if certain Java APIs are promised to application developers, then executable implementations of those Java APIs must be made available at runtime. Mr. Swetland’s focus on the “JVM” in part goes to this point.

261. The effect of these decisions and the benefit to Google is illustrated in Google’s Android OC Quarterly Review – Q4 2010 (by Andy Rubin et al.) (GOOGLE-01-00053552 at 563). The last two bullets highlight the importance of Android applications:

If we gave it away, how can we ensure we get to benefit from it?

Create policies that allow us to drive the standard

- Be the shepherds of the standard we created – we are in the lead because of our head start. Maintaining the pace will guarantee our lead.
- Do not develop in the open. Instead, make source code available after innovation is complete
- Lead device concept: Give early access to the software to partners who build and distribute devices to our specification (ie, Motorola and Verizon). They get a non-contractual time to market advantage and in return they align to our standard.
- We created the first app store for Android and it got critical mass quickly. The store now has value and partners want access to it because of the number of apps available.
- Own the ecosystem we enabled: Evolve the app store. Set the rules. Define developer monetization opportunity. Train developers on our APIs. Give developers one place where they get wide distribution. Provide a global opportunity & payment system. Help developers get distribution via revshare with operators. Extend app store to other devices and other market segments (ie, Google TV)

Takeaway: Provide incentives -- carrots rather than sticks


Google Confidential and Proprietary 12

B. Google Recognized that Sun Microsystems's (now Oracle's) Copyrighted Features Would Be the Key to Attracting Millions of Application Developers to Android and OEMs

262. Google has publicly stated on the Android Developer website (<http://developer.android.com/guide/practices/compatibility.html>) that “The goal of Android is to create a huge installed base for developers to take advantage of.” Why is that the case? It is because Google recognized that its adoption of Java would attract a large community of application developers for Android as demonstrated by the following email exchange between Android's key proponents:

August 17, 2007 Email
From: Alan Eustace (Google)
To: Andy Rubin (Google)

“I can tell you first hand that there are tens of thousands of Java developers who just can't wait to write mobile applications.... With Android, they are going to go absolutely nuts when they realize that they can write pretty much full-fledged java applications using 99% of the language.”

(8/17/2007 Email to Andy Rubin (GOOGLE-01-00029331).)

263. Google apparently understood that its attracting Java application developers would result in Google attracting applications for Android as demonstrated by the following email exchange between Android's key developers:

April 13, 2006 Email
 From: Andy Rubin (Google)
 To: Dan Bornstein (Google)
 ...
 "Java has very little fragmentation, and it's adoptable. If we play our cards right, we can also leverage not only existing developers, but applications as well."

(4/13/2006 Email from Andy Rubin to Dan Bornstein (GOOGLE-02-00111218).)

264. Google recognized that attracting application developers would be fundamental to the success of the Android application market and therefore to Android, as illustrated in the following statements from Andy Rubin from one of his depositions in this lawsuit:

"There is no purpose of building an open platform other than to attract third-party developers to it. So anything that we would do to jeopardize the support of third-party developers would be bad for the success of the platform." (4/5/2011 Rubin Dep. 91:19-23.)
 "Third-party developers contribute to the success of a platform by having their companies invest in the platform by basing their businesses on the platform. It was my intention to create an independent third-party developer ecosystem..." (4/5/2011 Rubin Dep. 24:21-25:2.)

As I understand it, the proposition in Mr. Rubin's testimony is that the whole point of Android is to attract application developers to it. As discussed above, the goal of attracting application developers is to make available a wide range of applications that would draw consumers to Android (and to buy Android devices). As I understand it, attracting more consumers to Android translates to more sales of Android devices (benefiting OEMs and carriers) and more ad revenue for Google.

265. The importance of Android applications and the importance of the surrounding ecosystem to application development is further emphasized in Google's Android OC Quarterly Review – Q4 2010 (by Andy Rubin et al.) (GOOGLE-01-00053552 at 563):

If we gave it away, how can we ensure we get to benefit from it?

Create policies that allow us to drive the standard

- Be the sheppards of the standard we created – we are in the lead because of our head start. Maintaining the pace will guarantee our lead.
- Do not develop in the open. Instead, make source code available after innovation is complete
- Lead device concept: Give early access to the software to partners who build and distribute devices to our specification (ie, Motorola and Verizon). They get a non-contractual time to market advantage and in return they align to our standard.
- We created the first app store for Android and it got critical mass quickly. The store now has value and partners want access to it because of the number of apps available.
- Own the ecosystem we enabled: Evolve the app store. Set the rules. Define developer monetization opportunity. Train developers on our APIs. Give developers one place where they get wide distribution. Provide a global opportunity & payment system. Help developers get distribution via revshare with operators. Extend app store to other devices and other market segments (ie, Google TV)

Takeaway: Provide incentives -- carrots rather than sticks


Google Confidential and Proprietary 12

266. Google also used Java to attract Carriers and OEMs. A January 31, 2006 Google presentation states that “Java **dominates** wireless industry” and is “Critical to our open source handset strategy.” (GOOGLE-14-00042244 at 246.) In addition, adopting Java “Dramatically accelerates our schedule” and “Forms an industry alliance to block MSFT.” (GOOGLE-14-00042244 at 246.) In other words, Google understood Java to be dominant and critical to Google’s strategy, including as a means to compete effectively with Microsoft.

267. Addressing both the importance of the developer community and appeal to carriers and the wireless industry, a Google presentation entitled “Android Open Handset Platform” from September 28, 2006 states:

“Fact ... 6M Java developers worldwide. Tools and documentation exist to support app development without the need to create a large developer services organization. There exist many legacy Java applications. The wireless industry has adopted Java, and the carriers require its support.

Strategy: Leverage Java for its existing base of developers.”

(GOOGLE-01-00025576 at 584.)

END GOOGLE ATTORNEYS’ EYES ONLY

C. Google Made Sun Microsystems's (now Oracle's) Copyrighted Features Necessary to Android

268. In adopting Java, Google took steps to maintain and provide developers with the core Java library that form the basis for Oracle's copyright infringement claim against Google as detailed in my report.

269. According to the Android Developer website (<http://developer.android.com/guide/basics/what-is-android.html>), "Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language."

270. These libraries are not only provided by Google to Android developers, but Google also mandates them on Android devices as set out in Google's Compatibility Definition Document. *See, e.g.*, Google's Android 2.2 to Compatibility Definition Document, available at <http://source.android.com/compatibility/downloads.html>:

"3.1. Managed API Compatibility

The managed (Dalvik-based) execution environment is the primary vehicle for Android applications. The Android application programming interface (API) is the set of Android platform interfaces exposed to applications running in the managed VM environment. Device implementations **MUST** provide complete implementations, including all documented behaviors, of any documented API exposed by the Android 2.1 SDK [Resources, 4].

Device implementations **MUST NOT** omit any managed APIs, alter API interfaces or signatures, deviate from the documented behavior, or include no-ops, except where specifically allowed by this Compatibility Definition."

271. Not only does Google mandate the core libraries, but Google prohibits modifications as elaborated further in Google's Compatibility Definition Document:

"Android follows the package and class namespace conventions defined by the Java programming language. To ensure compatibility with third-party applications, device implementers **MUST NOT** make any prohibited modifications (see below) to these package namespaces:

- java.*
- javax.*
- sun.*
- android.*
- com.android.*

Prohibited modifications include:

Device implementations **MUST NOT** modify the publicly exposed APIs on the Android platform by changing any method or class signatures, or by removing classes or class fields.

Device implementers **MAY** modify the underlying implementation of the APIs, but such modifications **MUST NOT** impact the stated behavior and Java-language signature of any publicly exposed APIs.

Device implementers **MUST NOT** add any publicly exposed elements (such as classes or interfaces, or fields or methods to existing classes or interfaces) to the APIs above.”

272. In other words, Google affirmatively chose to make the Java core libraries – which serve as the basis for Oracle’s copyright claim against Google – core to Android, mandated device implementers to provide the same core libraries on Android devices, and prohibited device implementers from modifying the same.

273. From the technical perspective, Google chose to mandate the features that are the subject of the copyrighted works asserted Oracle has asserted against Google in this lawsuit.

D. Conclusion

274. Below is a simplified structure of the information discussed above in relation to Oracle’s asserted copyrighted works:

Google targeted the mobile industry to increase its ad revenues and acquired Android, Inc. that produced Android

Google increases its ad revenues by having more consumers drawn to its Android mobile platform

What draws consumers is a wide range of appealing applications

The availability of applications requires application developers to build them

What has drawn application developers to Android rapidly is a familiar development environment that provides an ecosystem of tools and support

Java (including the copyrighted works asserted in this lawsuit) has drawn an extensive application developer community and already has an established rich ecosystem

Google adopted the Java copyrighted works at issue, such as the Java APIs, in Android to leverage Java’s extensive application development community and rich ecosystem

VII. CONCLUSION

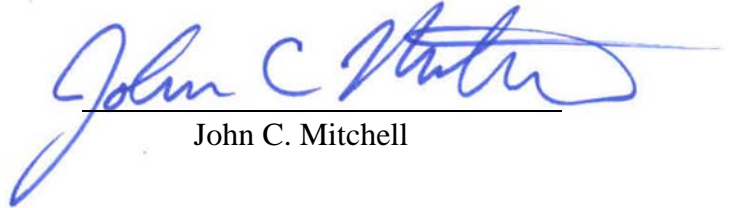
275. Based on my investigation, I have reached the following conclusions and will testify as to each.

276. The Java API specifications contain copyrightable expression.

277. Portions of the Android API specifications are substantially similar to the Java API specifications; the Android source code that implements the Android specifications contains these same elements; and other Android source code is substantially similar to Oracle's copyrighted source code or to decompiled Oracle object code.

278. The copyrighted features discussed in my report form a basis for customer demand for Android. Therefore, Android's success is in part due to the copyrighted works asserted in this case.

Dated: July 29, 2011



John C. Mitchell

Curriculum Vitae
John Clifford Mitchell
January 11, 2011

GENERAL RESEARCH INTERESTS

Computer security: access control, network protocols, privacy, software systems, and web security. Programming languages, type systems, object systems, and formal methods. Applications of mathematical logic to computer science

PERSONAL

Born December 20, 1955 in Palo Alto, CA.
U.S. citizen, married, two children.

ADDRESS

Home 845 Esplanada Way
Stanford, CA 94305

Office Department of Computer Science
Stanford University
Stanford, CA 94305-9045
(415) 723-8634

Net mitchell@cs.stanford.edu <http://www.stanford.edu/~jcm/>

EDUCATION

Ph.D. Computer Science, MIT, August, 1984. Thesis title: *Lambda Calculus Models of Typed Programming Languages*. Supervisor: A.R. Meyer.

S.M. Computer Science, MIT, January, 1982. Thesis title: *Axiomatic Definability and Completeness for Recursive Programs*. Supervisor: A.R. Meyer. Entered MIT September, 1980.

B.S. Mathematics with Distinction, Stanford University, June, 1978. Transferred September, 1976 from University of Wisconsin, Madison.

RESEARCH AND TEACHING POSITIONS

Present position:

Mary and Gordon Crary Family Professor in the School of Engineering,
Professor of Computer Science and (by courtesy) Electrical Engineering, Stanford University.
Associate Professor, 1990 – 1997, Assistant Professor, 1988 – 1990.

July–December, 1995:

Visiting Scientist and Program Co-organizer, *Semantics of Computation*, Newton Institute for Mathematical Sciences, Cambridge University.

September, 1984 to January, 1988 and Summer, 1983:

Member Technical Staff, Computing Science Research Center, AT&T Bell Laboratories.

Spring 1986:

Adjunct Assistant Professor, Dept. of Computer Science, New York University.

June, 1978 to August, 1980 and Summers 1976, 1977:

Research Engineer, University of Wisconsin Solar Energy Laboratory.

CONSULTING AND COMMERCIAL ACTIVITIES

PACid Group, LLC v. Apple, Inc. et. al./Fish & Richardson:

Technical expert for patent suit, June 2009 – September 2010, on behalf of defendants Intel, Marvell, Broadcom, and Atheros.

Network Appliance, Inc. v. Sun Microsystems, Inc./Weil, Gotshal & Manges:

Technical expert for patent suit, September 2009 – September 2010. Expert report and deposition.

Freyburger LLC v. Microsoft Corp./Weil, Gotshal & Manges:

Technical expert for patent suit, August 2009 – January 2010. Expert report.

Symbol Technologies and Wireless Valley Communications v. Aruba/Weil, Gotshal & Manges:

Technical expert for patent suit, June – October, 2009.

Cenzic, Inc.

Strategic Advisory Board, June 2008 – present.

Microsoft, Inc.

Technical security review of inter-frame messaging for Windows Live, March – May, 2008.

Trend v. Barracuda Networks/ McDermott Will & Emery:

Technical expert for patent suit, August 2007 – October 2008. Opening, rebuttal, and supplemental reports; deposition.

NovaShield, Inc.

Technical Advisory Board, January, 2007 – present.

Visto v. Microsoft/Manatt Phelps & Philips:

Technical expert for patent suit, June 2006 – February 2008. Opening, rebuttal, and supplemental reports; deposition.

Coverity, Inc.

Research and development consultant; technical advisor, October 2005 – June 2007.

PassMark Security, Inc.

Technical Advisory Board, October 2005 – May 2006 (acquired by RSA).

Southeast Texas Medical Associates v. Verisign, Inc./Arnold & Porter:

Technical expert for class-action suit, October 2005 – August 2007. Technical study and declaration.

Wi-Fi Alliance

Simple Configuration Security Review (wireless pairing and configuration of wireless networking devices), Sept – Dec, 2005.

Soverain Software v. Amazon.com and The Gap, Inc./Klarquist Sparkman:

Technical expert for patent suit, April – August 2005.

BTG v. Microsoft/Preston Gates Ellis:

Technical expert for patent suit, February – August 2005. Presented Markman hearing tutorial.

Trend Micro v. Fortinet/McDermott Will & Emery:

Technical expert, patent investigation filed with International Trade Commission, July 2004 – January 2006. Reports, deposition, and over 20 hours of testimony during 8 days of hearings.

Workshare/Coudert Brothers:

Study related to software and user interface copyright, Dec 2003 – Feb 2004.

Token-based authentication/Heller Ehrman White & McAuliffe/Dovell and Luner:

Technical expert for patent suit, Aug – Dec 2003. Expert report.

Network Caching Technology v. Novell . . . /Jones, Day, Reavis & Pogue:

Technical expert for patent suit, Apr 2002 – Sept 2003. Declarations.

Jupiter Media Metrix v. Nielsen Netratings/Brobeck Phleger & Harrison:

Technical expert for patent suit, Feb – May 2002.

InterTrust v. Microsoft/Klarquist Sparkman:

Technical expert for patent suit, Jan 2002 – April 2004. Report and deposition.

Indivos v. Biometric Access Corporation/Brobeck Phleger & Harrison:

Technical expert for patent suit, Jan – July 2002. Report and deposition.

Flyswat v. Third Voice/Heller Ehrman White & McAuliffe:

Technical expert for patent dispute, Sept – Oct 2000.

Blue Mountain Arts v. Lucidity/Mitchell Silberberg & Knupp:

Technical expert for contract dispute, June – Sept 2000.

Xerox PARC: Research on trust management, June – Oct 2000.

Trend Micro v. Network Associates/Townsend and Townsend and Crew:

Technical expert for patent suit. Reports, deposition, and testimony at jury trial. Sept 1999 – May 2000.

Kestrel Institute: Research on principles of computer security, logical methods for protocol analysis, Feb 1999 – June 2006.

Airtouch Corporation: Mathematical analysis of accounting methods related to international tax law, Dec 1997 – Aug 1998.

Neuron Data/Brobeck, Phleger & Harrison: Software copyright and contract dispute, October 1996 – January 1997.

Pure Software/Brobeck, Phleger & Harrison: Software patent cases, copyrights, contract disputes; July, 1994, January–March 1995 and June–August 1996.

Interval Research: Scripting languages for Internet applications, May–June 1995.

Kestrel Institute: Specification language and type/module system for prototyping language. February – May, 1992.

Hewlett-Packard Labs: ABEL project to develop typed, object-oriented programming language and methodology. May, 1988 – May, 1990.

RESEARCH CONTRACTS, GRANTS AND AWARDS

DARPA TYPed languages for Programming on Encrypted Data (TYPED), J Mitchell (PI), D Boneh.

DHHS Strategic health IT Advanced Research Projects on Security (SHARPS), April 2010 – March 2014. Chief Computer Scientist.

ONR MURI Botnet Attribution and Removal: From Axioms to Theory to Practice, July 2009 – May 2012. Georgia Tech: Wenke Lee (PI), Nick Feamster, Jon Giffin, David Dagon; Michigan: Kang Shin, Farnam Jahanian, Michael Bailey; Stanford: John Mitchell; UC Santa Barbara: Giovanni Vigna, Christopher Kruegel.

ONR Web Security and Mashups, March 2009 – Sept 2011. J. Mitchell (PI).

NSF Collaborative Research: CT-M: Privacy, Compliance and Information Risk in Complex Organizational Processes, Sept 2008 – Aug 2011. Anupam Datta (PI), John Mitchell (co-PI), Helen Nissenbaum (co-PI), Andre Scedrov (co-PI).

AFOSR MURI Collaborative policies and assured information sharing, June 2008 – May 2011. Dan Boneh (co-PI), Anupam Datta (co-PI), Joseph Hellerstein (co-PI), John Mitchell (PI), Helen Nissenbaum (co-PI), Tim Roughgarden (co-PI), Andre Scedrov (co-PI), Hovav Shacham (co-PI), Vitaly Shmatikov (co-PI), Dawn Song (co-PI), Brent Waters (consultant).

- DARPA Virtualized Execution Resulting in Network Infrastructures Enhancing Reliability (VERNIER).* June 2006–March 2008. Patrick Lincoln (SRI, PI), John Mitchell (co-PI).
- Dept Homeland Security SpoofGuard Anti-Phishing Technologies,* 2005–2007. Dan Boneh (PI), J. Mitchell (co-PI).
- NSF Science and Technology Center Team for Research in Ubiquitous Secure Technology (TRUST),* 2005–2010 (with possible extension to 2015). Shankar Sastry (UC Berkeley, PI), John C. Mitchell (co-PI), Michael Reiter (CMU, co-PI), Janos Sztipanovits (Vanderbilt, co-PI), and Steven Wicker (Cornell, co-PI).
- Dept Homeland Security System for Internet Relay Chat Underground Information Tracking (SIR-CUIT),* 2004–2005. John C. Mitchell (PI).
- NSF Cybertrust Program Collaborative Research: High-Fidelity Methods for Security Protocols,* 2004–2007. John C. Mitchell (PI), Danielle Micciancio (UCSD), Andre Scedrov (U Penn), Vitaly Shmatikov (U Texas).
- ONR University Research Initiative Trustworthy Infrastructure, Mechanisms and Experimentation for Diffuse Computing (TIME DC),* 2004–2006. Joan Feigenbaum (Yale), Joseph Y. Halpern (Cornell), Patrick D. Lincoln (SRI), John C. Mitchell (Stanford), Andre Scedrov (U Penn, PI), Steve Zdancewicz (U Penn).
- NSF Large ITR to Stanford University Sensitive Information in a Wired World,* 2003–2008. Dan Boneh (Stanford, PI), Joan Feigenbaum (Yale), Stephanie Forrest (Univ. New Mexico), Ravi Kannan (Yale), Hector Garcia-Molina (Stanford), John Mitchell (Stanford), Rajeev Motwani (Stanford), Helen Nissenbaum (NYU), Avi Silberschatz (Yale), Rebecca Wright (Stevens Institute).
- NSF Medium ITR to Brigham Young University Automated Trust Negotiation in Open Systems,* Fall 2003. Ninghui Li (Purdue), John Mitchell (Stanford), Kent Seamons (Brigham Young Univ., PI), Brian Tung (Univ. Southern California), William Winsborough (Network Associates Laboratories), Marianne Winslett (Univ. Illinois).
- DARPA/AFOSR MURI to Univ Illinois, Cooperative Control of Dynamical Mobile Agents via Network Protocols,* 2002–2005. Geir Dullerud (ME, UIUC, PI). Co-PIs: Jinane Abounadi (EECS, MIT), Francesco Bullo (Engr, UIUC), Eric Feron (Aero and Astro, MIT), Emilio Frazzoli (Aero and Astro, UIUC), P.R. Kumar (ECE, UIUC), Sanjay Lall (Aero and Astro, Stanford), Daniel Liberzon (ECE, UIUC), Nancy A. Lynch (EECS, MIT), John C. Mitchell (CS, Stanford), Sanjoy K. Mitter (EECS, MIT), Eytan Modiano (Aero and Astro, MIT), Bruce Reznick (Math, UIUC), Mahesh Viswanathan, (CS, UIUC).
- NSF Information Technology Research (ITR), 2001-2006.* Computational Logic Tools for Research and Education. David Dill (Stanford, PI), Zohar Manna (Stanford), John Mitchell (Stanford).
- ONR URI Program Software Quality and Infrastructure Protection for Diffuse Computing,* 2001–2004 and continuation 2004–2006. Joan Feigenbaum (Yale), Joseph Y. Halpern (Cornell),

Patrick D. Lincoln (SRI), John C. Mitchell (Stanford), Andre Scedrov (U Penn, PI), Jonathan M. Smith (U Penn).

DARPA99-33-034 Agile Management of Dynamic Collaboration, 2000-03. J. Mitchell (Principal Investigator), P.Lincoln (SRI, Co-PI), M. Baker (Stanford), D.Dill (Stanford), L. Gong (Java-Soft).

DERA, 1999-2000. Instrumentation and Checking Techniques Applied to Jini. J. Mitchell, Principal Investigator.

NSF-JAPAN Award, 1999-2002. Andre Scedrov, Principal Investigator.

DERA, 1998-1999. Instrumentation and Checking of Mobile Code. J. Mitchell, Principal Investigator.

ONR MURI Award Semantic Consistency in Information Exchange, 1997-2000 and continuation 2000-02. J. Mitchell (Principal Investigator), S. Kannan, I. Lee and A. Scedrov (Univ. Pennsylvania), R. Rubinfeld (Cornell), P. Lincoln (SRI), C. Dwork (IBM).

NSF Grant CCR-9629754 Object Systems: Programming Languages and Software Security, 1996-99. J. Mitchell, Principal Investigator.

TRW Foundation Grant to study programming languages and system design languages, 1994. J. Mitchell, Principal Investigator.

NSF Grant Programming Language Analysis and Design, 1993-96. J. Mitchell, Principal Investigator.

NSF Presidential Young Investigator Award, 1988-93. Industrial funding provided by AT&T, Digital Equipment Corporation, Mitsubishi Corporation, the Powell Foundation and Xerox Corporation.

DARPA/ISTO BAA 89-08 Project to develop Common Prototyping Language and design accompanying Common Prototyping System, 1989-90 and continuations through 1995. Principal investigators: David Luckham (Stanford) and Frank Belz (TRW).

NSF grant Research in Programming Language Structures: Types and Concurrency, 1989-91. J. Mitchell and V. Pratt, Principal Investigators.

NSF-INRIA grant for US-France collaboration, 1989-91. Val Breazu-Tannen, Carl Gunter principal investigators. Principal INRIA contacts: Gérard Huet (Paris) and Gilles Kahn (Sophia-Antipolis).

CNR (Italy) grant for Stanford-Italy collaboration, 1989-91. J. Barwise, S. Feferman, J. Mitchell (Stanford), M. Dezani (Turin), G. Longo (Pisa) principal investigators.

HONORS AND AWARDS

Dean's Award for Industry Education Innovation, 2009.
 Fellow of the Association for Computing Machinery, elected 2008.
 Computerworld New Horizons Award, 2006.
 The Mary and Gordon Crary Family Professorship (2004–present),
 Director's Award, U.S. Secret Service (2003),
 Wallace F. and Lucille M. Davis Faculty Scholar (1989–91),
 NSF Presidential Young Investigator Award (1988–93),
 IBM Graduate Fellowship (1983–84),
 NSF Graduate Fellowship (1980–83),
 Graduation with Distinction (Stanford 1978),
 Juliet Knopp Lockwood Honors Scholarship (Stanford 1977),
 Invitation to H^* honors mathematics program (U.W. 1975).

PROFESSIONAL ACTIVITIES

Service to Professional Organizations:

ACM-Infosys Award, Selection Committee, 2007 – 2010,
Max Planck Institute for Software Systems, Advisory Board, 2008(?) – present
Computing Community Consortium, Council member, 2011 – present,
IFIP Technical Committee TC1, Foundations of Computer Science, 2004 – 2010,
NSF National Cyber-Physical Systems Virtual Organization (CPS-VO), Academic Executive Board, 2010 – present.

Series Editor:

Electronic Notes in Theoretical Computer Science, 1995 – 2000,
Springer Lecture Notes in Computer Science, 2004 – present.
Information Security and Cryptography (Book series, Springer-Verlag) 2007 – present.

Journal Advisory Board:

J. of Privacy and Confidentiality, 2006 – present.

Editor-in-Chief: Journal of Computer Security, 2010 – present.

Journal Editorial Boards:

ACM Trans. Computational Logic, 2003 – present,
ACM Trans. Programming Languages and Systems (TOPLAS), 1993 – 1996,
ACM Transactions on Information and System Security (TISSEC), 2006 – present,
Chicago J. Theoretical Computer Science, 1994 – 2000,
Information and Computation, 1987 – 2000,
J. Assoc. Computing Machinery (JACM), 1995 – 2000,
J. Computer Security, 2000 – present,
J. Functional Programming, 1989 – 2010,
Mathematical Structures in Computer Science, 1989 – 2000,

SIAM J. Computing, 1998 – 2004,
Theory and Practice of Object Systems, 1994 – 2000.

Special Issue Editor:

Info. and Computation: 1990 IEEE Logic in Comp. Sci. Conference.

Conference General Chair:

IEEE Symp. on Logic in Computer Science, 1998-2001. Organizing committee chair - responsible for program committees and scientific direction of conference.

Conference Organizing Committees:

ACM Conference on Computer Security (CCS), 2011 – present,
 IEEE Symp. on Logic in Computer Science, 1990 – present,
 Category Theory and Computer Science, 1990 – 1996.
 Third Symp. Theor. Aspects of Computer Software (Advisory Committee) 1997.
 Foundations of Computer Security, 2002 – present. Founder and Organizing Committee Chair, 2002 – 2004.
 IEEE Symp. on Computer Security Foundations, 2008 – present.

Conference Program Chair:

IEEE Symp. on Logic in Computer Science, 1990,
 Second Symp. on Theoretical Aspects of Computer Software, Sendai, Japan, 1994 (co-chair),
 Advances in Types Systems for Computing, Newton Institute, Cambridge, 1995,
 ACM Symp. on Principles of Programming Languages, Portland Oregon, 2002,
 IFIP Symp. on Theoretical Computer Science (TCS 2004), Track B, Toulouse France, 2004.
 Formal Methods in Security Engineering (FMSE), 2005.
 Joint iTrust and PST conf. on Privacy, Trust Management, and Security (IFIPTM 2008)
 IEEE Symp. Computer Security Foundations (CSF), 2009.

Conference Program Committees:

ACM Conf. Functional Programming and Computer Architecture (FPCA), 1989 and 1995.
 ACM Conf. on Object-Oriented Programming (OOPSLA), 1988, 1992 and 1995.
 ACM Symp. Computer and Communication Security (CCS), 2004, 2005, 2006, 2007, 2008.
 ACM Symp. Principles of Programming Languages (POPL), 1989, 1991, 2000, and 2002.
 ACM Workshop on Formal Methods in Security Engineering (FMSE), 2004.
 ACM Workshop on ML, 1992.
 ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS), 2010, Annual Asian Computing Science Conference (Asian'06), 2006.
 Association for Symbolic Logic Annual Meeting, 2005.
 Category Theory and Computer Science (CTCS), 1991, 1993, 1995 and 1997.
 Conference on Automated Deduction (CADE), 2003.
 Conference on Automated Verification (CAV), 2000.
 Conference on Concurrency Theory (CONCUR), 2005.
 Conference on Deontic Logic in Computer Science (DEON), 2008.
 Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA),

2008.

Conference on Email and Anti-Spam (CEAS), 2005.

Conference on Security and Cryptography for Networks (SCN), 2010,

European Symposium On Programming (ESOP), 2002, 2003, 2007.

European Symposium On Research In Computer Security (ESORICS), 2000, 2009, 2010,

Foundations of Object-Oriented Languages (FOOL), 1995, 1996, 1997, 2005.

Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2003.

Fundamentals of Computation Theory (FCT), 1999.

ICST Conference on Security and Privacy in Communication Networks (SecureComm), 2009.

Information Security Conference (ISC), 2009.

Int'l Colloquium on Automata, Languages, and Programming (ICALP), 2006 (Track C).

Int'l Conference on Availability, Reliability and Security (ARES), 2008,

IEEE Computer Security Foundations Workshop (CSFW), 1999, 2002, 2004, 2005, 2006.

IEEE Symposium on Computer Security Foundations (CSF), 2007, 2008, 2009.

IEEE Symp. on Foundations of Computer Science (FOCS), 1988 and 1991.

IEEE Symp. on Logic in Computer Science (LICS), 1986, 1988, 1990, 1996, 2008, 2010.

IEEE Symp. on Security and Privacy, 1999, 2004, 2009.

IFIP International Conference on Theoretical Computer Science (TCS), 2002 and 2004.

Mathematical Foundations of Programming Language Semantics (MFPS), 1991, 1993, 1995.

Public Key Infrastructure (PKI R&D), 2005.

Techniques of Object-Oriented Languages and Systems (TOOLS), 1992 and 1993.

Theory of Cryptography Conference (TCC), 2006.

Typed Lambda Calculi and Applications (TLCA), 1993.

Web 2.0 Security and Privacy (W2SP), 2007, 2008, 2009.

Workshop on Advances in Policy Enforcement, 2008.

Workshop on Formal Methods and Computer Security (FMCS), 2000.

Workshop on Foundations of Computer Security (FCS), 2003, 2005.

Workshop on Foundations of Global Computing (FGC), 2003.

Workshop on Issues in the Theory of Security (WITS), 2005, 2010.

Workshop on Logic, Language Information, and Computation (WoLLIC), 2007.

Workshop and Program Organization:

- DIMACS Workshop on Security Protocols, June 7–9, 2004. Co-organizer with R. Canetti (IBM).
- Workshop on Programming Languages and Security, DEC Systems Research Center, Oct 30–31, 1997. With M. Abadi (DEC, principal organizer), B. Bershad (U. Washington), E. Felton (Princeton), L. Gong (JavaSoft).
- Working group on Software Engineering and Programming Languages, CRA Meeting on Strategic Directions in Computing, on the occasion of the 50th anniversary of the ACM. Boston, June 14–15, 1996.

Report: Gunter, C., Mitchell, J.C. and Notkin, D., Strategic Directions in Software Engineering and Programming Languages, *ACM Computing Surveys*, Vol 28A, No 4, December 1996.

- Workshop on Software Engineering and Programming Languages, sponsored by ARO and NSF, Boston, June 12–13, 1996. Co-organizer and program co-chair (with D. Notkin).
- Co-Organizer and meeting chair, ARPA-NSF workshop on Foundational Studies for Software Engineering, Stanford, Sept 6–7, 1995.
- Co-organizer, with S. Abramsky (Imperial), G. Kahn (INRIA) and A. Pitts (Cambridge), “Programme on Semantics of Computation,” Isaac Newton Institute of Mathematical Sciences, University of Cambridge, U.K., July–December, 1995. This six-month program on involved approximately 65 visitors from North America, Europe and Japan, as well as hundreds of short-term participants in 10 special conferences and workshops at the Institute.
- Founder and initial organizer, Foundations of Object-Oriented Languages (FOOL), annual workshop since 1993.
- Co-organizer, with D. Liddle (Interval), Stanford Workshop on Software Engineering, Sept 12–13, 1993.
- A founding organizer of “North-American Jumelage,” annual workshop on programming language theory and logic in computer science, 1990–1995. Hosted first meeting at Stanford, 1990.

Column: *Sigact News* Logic Column, 1991 – 1997.

Regular Published Reviews:

Mathematical Reviews, Amer. Math. Society, 1989 – 1993

Zentralblatt für Mathematik/Mathematical Abstracts, Springer-Verlag, 1990 – 1993.

Member: IFIP Technical Committee TC1, IFIP Working Group 1.7, *Foundations of Security Analysis and Design* (Invited Charter Member), Assoc. Computing Machinery, Assoc. Symbolic Logic, European Assoc. for Theoretical Computer Science. Former member of IFIP Working Group 2.8, *Functional Programming* (Invited Charter Member).

INVITED LECTURES

1. Distinguished Lecture, Department of Computer Science, University of Wisconsin, April 2010.
2. Kanellakis Lecture (Departmental Distinguished Lecture), Brown University, December 2009.
3. Computational and Symbolic Proofs of Security, Spring School and French-Japanese collaboration workshop, April 2009.

4. Usenix Security Symposium, July 2008.
5. Brainstorms, a quarterly Stanford lecture series for the general public, Autumn 2007.
6. Marktoberdorf Summer School, 2007.
7. Mathematical Foundations of Program Semantics (MFPS), New Orleans, April, 2007.
8. Asian 2006, Tokyo, December, 2006.
9. Logic and Computational Complexity (LCC), Seattle, August, 2006.
10. International Colloquium on Automata, Languages and Programming (ICALP), Lisbon, July, 2005. First year of *Track C: Security and Cryptography Foundations*.
11. Second Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA), Lisbon, July, 2005.
12. TCS Excellence in Computer Science Week (TECS Week), Pune, India, January 4–8, 2005. Series of lectures on security analysis of network protocols.
13. ACM Workshop on Formal Methods in Security Engineering (FMSE), Washington D.C., October, 2004.
14. Usenix Security Symposium, San Francisco, August, 2002.
15. Rewriting Techniques and Applications (RTA), Copenhagen, July, 2002.
16. Mathematical Foundations of Programming Semantics (MFPS), New Orleans, March, 2002.
17. IEEE Symp. Logic in Computer Science (LICS), Boston, June 2001.
18. European Symposium on Programming (ESOP), Genoa, Italy, April 2001.
19. Association for Symbolic Logic (ASL) Annual Meeting, Philadelphia, March 2001.
20. ACM Symp. Principles of Programming Languages (POPL), London, U.K., January 2001.
21. Computer Science Logic (CSL '98), Brno (Czech Republic), August 24–28, 1998.
22. Marktoberdorf Summer School, 1998. Five lectures on security, network protocols and formal methods.
23. Conference on Computer-Aided Verification (CAV '98), June 28 – July 2, 1998, Vancouver, British Columbia.
24. Seventh CSLI Workshop on Logic, Language and Computation, May 29 – 31, 1998, Stanford University.
25. Linear '98, CIRM Luminy, Marseille, April 6–9, 1998.

26. ACM Workshop on Functional and Object-oriented Programming, July 2-7, 1997, Jadwisin (Poland)
27. Marktoberdorf Summer School, 1996. Four lectures on type systems and object-oriented programming.
28. Linear '96, Tokyo, Japan. March 28–April 2, 1996. Lecture series on Decision and Optimization Problems in Linear Logic, with P. Lincoln and A. Scedrov.
29. Workshops and conferences associated with Isaac Newton Institute special program on Semantics of Computation:
 - (a) Themes in the Semantics of Computation (org. S. Abramsky), July 17–21, 1995.
 - (b) Linear Logic and Applications (org. G. Bierman), Oct 16–18, 1995.
 - (c) Higher-order Techniques in Operational Semantics (org. A. Gordon) Oct 28-30, 1995.
 - (d) Games, Processes and Logic (org. S. Abramsky), Nov 6–10, 1995.
30. Fundamentals of Computation Theory (FCT'95) Dresden, Germany, August 22–25, 1995.
31. EATCS Summer School on Typed Lambda Calculus and Functional Programming, Udine, Italy, September 20–30, 1994.
32. Second Symp. Theoretical Aspects of Computer Software, Sendai, Japan, April 18–21, 1994.
33. Invited tutorial, Techniques for Object-Oriented Languages and Systems (TOOLS) Conference, Santa Barbara. August, 1992.
34. Second Montreal Workshop on Programming Language Theory, Montreal, Quebec. December, 1991.
35. Sixth Workshop on Mathematical Foundations of Programming Semantics, Kingston, Ontario. May, 1990.
36. Japan Society for Software Science and Technology Annual Workshop on Object-Oriented Programming, Hakone, Japan, March, 1990.
37. MSRI Workshop on Logic from Computer Science, Berkeley, Nov., 1989.
38. Summer Conference on Category Theory and Computer Science (third biennial conference), Manchester, U.K., Sept., 1989.
39. IBM Distinguished Lecture Series on Semantics, April, 1989.
40. Special session on the semantics of inheritance, Fifth Workshop on Mathematical Foundations of Programming Semantics, New Orleans, March, 1989.
41. Annual Meeting of the Assoc. for Symbolic Logic, Los Angeles CA, January, 1989.

42. *Logic Colloquium '88*, Int'l meeting of the Assoc. for Symbolic Logic, Padova Italy, August, 1988.
43. CMU Workshop on the Semantics of Lambda Calculus and Category Theory, April, 1988.
44. Institute on Logical Foundations of Functional Programming, Univ. Texas Year of Programming, June, 1987. Organizer: G. Huet.
45. Institute on Encapsulation, Modularization and Reusability, Univ. Texas Year of Programming, April, 1987. Organizer: D. Gries.
46. Mid-Atlantic Mathematical Logic Seminar, Philadelphia, PA, February, 1987.

SELECTED PANELS

1. Computer Security Foundations Symposium: the Next 20 Years, CSF 2007, Venice, 2007.
2. International Association of Privacy Professionals (IAPP) Privacy Summit, 2004.
3. Mobicom '03, Panel on Computer Security, September 2003.
4. KQED Forum (radio program) with host Michael Krasny, Computer Security, September 9, 2003, with guests Joe Anastasi (Deloitte and Touche), Lee Tien (Electronic Frontier Foundation), Doug Howard (Counterpane Internet Security), Robert Rodriguez (US Secret Service). Interviewed in 2004 for NPR *Morning Edition*.
5. Invited panelist, Workshop on Challenges for Theoretical Computer Science (Organizers: David Johnson, Christos Papadimitriou, Avi Wigderson, Mihalis Yannakakis), Portland, OR, May 20, 2000.
6. Panelist, Software Engineering and Programming Languages, *Sigsoft '96*, ACM Symposium on Foundations of Software Engineering, 1996.

PUBLICATIONS

Books

1. J.C. Mitchell, *Concepts in Programming Languages*, Cambridge University Press, 2002, 529 pages.
2. J.C. Mitchell, *Foundations for Programming Languages*, MIT Press, 1996, 846 pages.
3. C.A. Gunter and J.C. Mitchell (eds.), *Theoretical Aspects of Object-Oriented Programming*, MIT Press, 1994, 548 pages.

Invited contributions to books

1. A. Roy, A. Datta, A. Derek, J.C. Mitchell, and J-P Seifert, Secrecy Analysis in Protocol Composition Logic. In *Formal Logical Methods for System Security and Correctness*, IOS Press, 2008, in press. (Based on presentations at Summer School 2007, Formal Logical Methods for System Security and Correctness, Marktoberdorf, Germany.)
2. A. Datta, A. Derrick, J.C. Mitchell and A. Roy, Protocol composition logic (PCL), in *Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin*, ed. L. Cardelli, M. Fiore and G. Winskel, Electronic Notes in Theoretical Computer Science, 2007.
3. Phishing and Countermeasures, by M. Jacobsson and S. Myers, Wiley, 2007. Contributed Chapter 12, "Protecting browser state." Joint work with Collin Jackson, Andrew Bortz, and Dan Boneh.
4. N.A. Durgin and J.C. Mitchell, Analysis of Security Protocols. In *Calculational System Design*, ed. M. Broy and R. Steinbruggen, IOS Press, 1999, pages 369–395. (Based on presentations at NATO Advanced Study Institute on Mathematical Methods in Program Development, Marktoberdorf, Germany, 1998.)
5. Mitchell, J.C., Hoang, M.K. and Howard, B., Labeling Techniques for Typed Fixed-point Operators. In *Higher-order Operational Techniques in Semantics*, ed. A.D. Gordon and A.M. Pitts, Publication of the Newton Institute, Cambridge Univ. Press, 1998, pages 137–174.
6. Fisher, K. and Mitchell, J.C., On the relationship between classes, objects and data abstraction. In *Mathematical Methods in Program Development*, Springer-Verlag NATO ASI series F (Computer and System Sciences), vol. 158, 1997, pages 371–408. (Proc. NATO Advanced Study Institute on Mathematical Methods in Program Development, Marktoberdorf, Germany, July 30 – August 11, 1996.)
7. Lincoln, P.D., J.C. Mitchell and A. Scedrov, Stochastic interaction and linear logic. In *Advances in Linear Logic*, ed. J.-Y. Girard, Y. Lafont and L. Regnier, London Mathematical Society Lecture Notes, Volume 222, Cambridge University Press, 1995, pages 147–166. (Refereed)
8. Mitchell, J.C., On the equivalence of data representations. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, ed. V. Lifschitz, Academic Press, 1991, pages 305–330.
9. Kanellakis, P.C., Mairson, H.G. and Mitchell, J.C., Unification and ML type reconstruction. In *Computational Logic: Essays in Honor of Alan Robinson*, ed. J.-L. Lassez and G.D. Plotkin, MIT Press, 1991, pages 444–478.
10. Mitchell, J.C., Type Systems for Programming Languages. In *Handbook of Theoretical Computer Science*, ed. J. van Leeuwen, North-Holland, 1990, pages 366–458.
11. In *Logical Foundations of Functional Programming*, ed. Gérard Huet, Addison-Wesley, 1990:
 - (a) Mitchell, J.C., Polymorphic type inference and containment, pages 153–194,

- (b) Mitchell, J.C., A type inference approach to reduction properties and semantics of polymorphic expressions (summary), pages 195–212,
- (c) Bruce, K.B., Meyer, A.R. and Mitchell, J.C., The semantics of second-order lambda calculus, pages 213–272,
- (d) Meyer, A.R., Mitchell, J.C., Moggi, E. and Statman, R., Empty types in polymorphic lambda calculus, pages 273–284.

Refereed Articles in Archival Journals

1. A. Barth, C. Jackson, and J.C. Mitchell, Securing Frame Communication in Browsers, Communications of the ACM (CACM), vol. 52, no. 6, June 2009, pages 83-91. (Invited CACM Research Highlight)
2. C. He, M. Sundararajan, A. Datta, A. Derek, J. C. Mitchell, A Modular Correctness Proof of TLS and IEEE 802.11i, ACM Transactions on Information and System Security (in review).
3. R. Kuesters, A. Datta, J.C. Mitchell, and A. Ramanathan, On the Relationships Between Notions of Simulation-Based Security, *J. Cryptology*, 21(4):492546, 2008.
4. M. Backes, A. Datta, A. Derek, J. C. Mitchell, M. Turuani, Compositional Analysis of Contract-Signing Protocols, *Theor. Comput. Sci.* 367(1-2): 33-56 (2006)
5. N. Li and J.C. Mitchell, Understanding SPKI/SDSI using first-order logic. *Int. J. Inf. Sec.* 5(1): 48-64 (2006)
6. J.C. Mitchell, A. Ramanathan, A. Scedrov, V. Teague, A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols, *Theoretical Computer Science* 353 (2006) 118–164.
7. A. Datta, A. Derek, J. C. Mitchell, D. Pavlovic, A Derivation System and Compositional Logic for Security Protocols, *Journal of Computer Security (Special Issue of Selected Papers from CSFW-16)*, Vol. 13, 2005, pages 423–482.
8. I. Cervesato, N.D. Durgin, P.D. Lincoln, J.C. Mitchell, A. Scedrov, A Revised Comparison Between Strand Spaces and Multiset Rewriting for Security Protocol Analysis, *Journal of Computer Security*, volume 13, issue 2, 2005, pages 265–316.
9. N. Li, J.C. Mitchell, and W.H. Winsborough, Beyond Proof-of-compliance: Security Analysis in Trust Management, *J. ACM*, Vol. 52, 2005, pages 474–514.
10. N. Li and J.C. Mitchell, Understanding SPKI/SDSI Using First-Order Logic, *International Journal of Information Security*, Nov 2005. www.springerlink.com/link.asp?id=107927
11. R. Chadha, J.C. Mitchell, A. Scedrov, V. Shmatikov. Contract signing, optimism, and advantage. *J. Log. Algebr. Program.* 64(2): 189-218 (2005)

12. A. Chander, D. Dean, and J. Mitchell, Reconstructing Trust Management, *Journal of Computer Security*, vol 12, number 1, 2004, pages 131–164.
13. N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov, Multiset rewriting and the complexity of bounded security protocols, *Journal of Computer Security*, vol 12, number 1, 2004, pages 677–722.
14. N.A. Durgin, J.C. Mitchell, D. Pavlovic, A compositional logic for proving security properties of protocols, *Journal of Computer Security*, vol 11, number 4, 2003, pages 677–721.
15. S.N. Freund and J.C. Mitchell, A Type System for the Java Bytecode Language and Verifier. *Journal of Automated Reasoning*, volume 30 (3-4):271–321, 2003.
16. N. Li, W. Winsborough, J.C. Mitchell, Distributed Credential Chain Discovery in Trust Management, *J. Computer Security*, vol 11, number 1, 2003, pages 35–86.
17. Shmatikov, V. and Mitchell, J.C., Finite-State Analysis of Two Contract Signing Protocols, *Theoretical Computer Science* 283, June 2002, pages 419–450.
18. Freund, S.N. and Mitchell, J.C., A Type System for Object Initialization in the Java Bytecode Language, *ACM Trans. on Programming Languages and Systems*, 21, November 1999, pages 1196–1250.
19. Harper, R. and Mitchell, J.C., Parametricity and variants of Girard’s J operator, *Information Processing Letters* 70, 1999, pages 1–5.
20. Lincoln, P.D., Mitchell, J.C. and Scedrov, A., Optimization complexity of linear logic proof games, *Theoretical Computer Science*, Special Issues on Linear Logic (accepted for publication).
21. Fisher, K. and Mitchell, J.C., On the relationship between classes, objects and data abstraction, *Theory and Practice of Object Systems*, Volume 4, number 1, 1998, pages 3–25. Invited paper for Special Issue on Third Workshop on Foundations of Object-Oriented Languages (held 1996).
22. Lincoln, P.D., Mitchell, J.C. and Scedrov, A., Linear Logic Proof Games and Optimization, *Bulletin of Symbolic Logic*, 2,3 (1996) 322-338.
23. Fisher, K., and Mitchell, J.C., The Development of Type Systems for Object-Oriented Languages, *Theory and Practice of Object Systems* 1,3 (1996) 189–220.
24. Mitchell, J. and Viswanathan, R., Standard ML-NJ weak polymorphism and imperative constructs, *Information and Computation* 127, 2 (1996) 102-116. Invited for special issue from IEEE Symp. Logic in Computer Science, 1993.
25. Fisher, K., Honsell, F. and Mitchell, J.C., A lambda calculus of objects and method specialization, *Nordic J. Computing* (formerly *BIT*) 1, 1 (1994) 3–37. Invited paper.

26. Mitchell, J.C., On abstraction and the expressive power of programming languages. *Science of Computer Programming* 212 (1993) 141–163. Invited for special issue of papers from Symp. Theor. Aspects of Computer Software, Sendai, Japan, 1991.
27. Cardelli, L., Martini, S., Mitchell, J. and Scedrov, A., An extension of System F with subtyping, *Information and Computation* 109 (1994) 4–56. Invited for special issue of papers from Symp. Theor. Aspects of Computer Software, Sendai, Japan, 1991.
28. Jategaonkar, L. and Mitchell, J.C., Type inference with extended pattern matching and subtypes, *Fund. Informaticae* 19 (1993) 127–166. Invited for special issue on type systems, ed. J. Tiuryn.
29. Lincoln, P., Mitchell, J.C., Scedrov, A. and Shankar, N., Decision Problems for Propositional Linear Logic, *Ann. Pure and Applied Logic* 56 (1992) 239–311.
30. Harper, R. and Mitchell, J.C., The type structure of Standard ML, *ACM Trans. Programming Languages and Systems*, 15, 2 (1993) 211–252.
31. Mitchell, J.C., Type inference with simple subtypes, *J. Functional Programming* 1, 3 (1991) 245–286.
32. Cardelli, L. and Mitchell, J.C., Operations on records, *Mathematical Structures in Computer Science* 1 (1991) 3–48.
33. Mitchell, J.C. and Moggi, E., Kripke-style models for typed lambda calculus, *Ann. Pure and Applied Logic* 51 (1991) 99–124.
34. Bruce, K.B., Meyer, A.R. and Mitchell, J.C., The semantics of second-order lambda calculus, *Information and Computation* 85,1 (1990) 76–134.
35. Mitchell, J.C. and Plotkin, G.D., Abstract types have existential type, *ACM Trans. Programming Languages and Systems*, 10, 3 (1988) 470–502.
36. Mitchell, J.C., Polymorphic type inference and containment, *Information and Computation* 76, 2/3 (1988) 211–249.
37. Dwork, C., Kanellakis, P.C. and Mitchell, J.C., On the sequential nature of unification, *J. Logic Programming* 1 (1984) 35–50.
38. Mitchell, J.C., The implication problem for functional and inclusion dependencies, *Information and Control* 56 (1983) 154–173. Abstract in *Zentralblatt für Mathematik* 539.68090 (1985).
39. Meyer, A.R. and Mitchell, J.C., Termination assertions for recursive programs: completeness and axiomatic definability, *Information and Control* 56 (1983) 112–138. Summary in *Zentralblatt für Mathematik* 537.68034 (1985).
40. Braun, J. E. and J.C. Mitchell, Solar Geometry for Fixed and Tracking Surfaces, *Solar Energy* 31, 5 (1983) 439–444.

41. Mitchell, J.C., Theilacker, J.C. and Klein, S.A., Calculation of monthly average collector operating time and parasitic energy requirements. *Solar Energy Journal* 26, 6 (1981).

Conference Publications (competitive selection based on 10 page technical summary)

1. D. Akhawe, A. Barth, P. Lam, J.C. Mitchell and D. Song, Towards a formal foundation of Web security, Proc. IEEE Symposium on Computer Security Foundations, July 2010.
2. S. Maffeis, J.C. Mitchell and A. Taly, Object Capabilities and Isolation of Untrusted Web Applications, Proc. IEEE Symposium on Security and Privacy, May 2010.
3. J. Bau, E. Bursztein, D. Gupta, J.C. Mitchell State of the Art: Automated Black-Box Web Application Vulnerability Testing, Proc. IEEE Symposium on Security and Privacy, May 2010.
4. E. Bursztein, S. Bethard, J.C. Mitchell, D. Jurafsky, C. Fabry How Good are Humans at Solving CAPTCHAs? A Large Scale Evaluation Proc. IEEE Symposium on Security and Privacy, May 2010.
5. J. Bau and J.C. Mitchell, A Security Evaluation of DNSSEC with NSEC3, Network and Distributed Systems Security (NDSS), 2010.
6. A. Barth, B. Rubinstein, M. Sundararajan, J.C. Mitchell, D. Song, and P.L. Bartlett, A Learning-Based Approach to Reactive Security Proc. of the 14th International Conference on Financial Cryptography and Data Security (FC 2010), 2010.
7. E. Bursztein, J.C. Mitchell, Using Strategy Objectives for Network Security Analysis, In-scrypt'09, December 2009.
8. E. Bursztein, P.E. Lam, J.C. Mitchell, TrackBack Spam: Abuse and Prevention, Proc. ACM Cloud Computing Security Workshop, Chicago, November, 2009.
9. S. Maffeis, J.C. Mitchell and A. Taly, Isolating JavaScript with Filters, Rewriting, and Wrappers, 14th European Symposium on Research in Computer Security (ESORICS), Saint Malo, France, September 21-25, 2009.
10. P.F. Lam, J.C. Mitchell, and S. Sundaram, A Formalization of HIPAA for a Medical Messaging System, 6th International Conference on Trust, Privacy & Security in Digital Business (TrustBus'09), 2009.
11. T. Hinrichs, N. Gude. M. Casado, J.C. Mitchell, S. Shenker, Practical Declarative Network Management, ACM SIGCOMM Workshop: Research on Enterprise Networking (WREN 2009), 2009.
12. S. Maffeis, J.C. Mitchell and A. Taly, Run-Time Enforcement of Secure JavaScript Subsets, Web 2.0 Security and Privacy (W2SP'09), 2009.

13. S. Maffeis, (J.C. Mitchell) and A. Taly, Language-Based Isolation of Untrusted JavaScript, IEEE Symp. Computer Security Foundations (CSF'09), 2009.
14. S. Maffeis, J.C. Mitchell and A. Taly, An Operational Semantics for Javascript, ASIAN Symposium on Programming Languages and Systems (APLAS), 2008.
15. A. Barth, C. Jackson, and J.C. Mitchell, Robust Defenses for Cross-Site Request Forgery, Proc. of the 15th ACM Conference on Computer and Communications Security (CCS 2008), 2008.
16. E. Stinson, J.C. Mitchell, Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods, 2nd USENIX Workshop on Offensive Technologies (WOOT '08), 2008.
17. L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, J.C. Mitchell, A Layered Architecture for Detecting Malicious Behaviors 11th Int'l Symp. Recent Advances In Intrusion Detection (RAID), 2008.
18. A. Barth, C. Jackson, and J.C. Mitchell, Securing Frame Communication in Browsers, Proc. of the 17th USENIX Security Symposium. (USENIX Security 2008), 2008.
19. J.C. Mitchell, A. Roy, P. Rowe, A. Scedrov, Analysis of EAP-GPSK Authentication Protocol, Applied Cryptography and Network Security, 2008.
20. A. Miura-Ko, B. Yolken, J.C. Mitchell and N. Bambos, Security decision-making among interdependent organizations, IEEE Computer Security Foundations (CSF), 2008.
21. A. Roy, A. Datta, J.C. Mitchell, Formal Proofs of Cryptographic Security of Diffie-Hellman based Protocols, Proc. Symposium On Trustworthy Global Computing, Springer LNCS, November 2007.
22. A. Roy, A. Datta, A. Derek, J.C. Mitchell, Inductive Proofs of Computational Secrecy, Proc. 12th European Symposium On Research In Computer Security, September, 2007.
23. E. Stinson and J.C. Mitchell, Characterizing Bots Remote Control Behavior, 4th GI Int'l Conf. on Detection of Intrusions & Malware, and Vulnerability Assessment (DIMVA), Lucerne, Switzerland, July, 2007.
24. C. Jackson, D. Boneh, and J.C. Mitchell, Transaction Generators: Rootkits for the Web, 2nd USENIX Workshop on Hot Topics in Security (HotSec '07). (Workshop paper)
25. A. Barth, J.C. Mitchell, A. Datta and S. Sundaram, Privacy and Utility in Business Processes, 20th IEEE Computer Security Foundations Symposium (CSF 20), Venice, July, 2007.
26. A. Roy, A. Datta, A. Derek and J.C. Mitchell. Inductive Trace Properties Imply Computational Security, 7th International Workshop on Issues in the Theory of Security (WITS'07), Braga, Portugal, March, 2007.

27. A. Roy, A. Datta, A. Derek, J.C. Mitchell, J.-P. Seifert, Secrecy Analysis in Protocol Composition Logic, 11th Annual Asian Computing Science Conference (ASIAN'06), Tokyo, December, 2006.
28. A. Barth, J.C. Mitchell, Managing Digital Rights using Linear Logic, Proceedings 21st Annual IEEE Symposium on Logic in Computer Science, Seattle, August, 2006.
29. A. Datta, A. Derek, J.C. Mitchell, B. Warinschi, Key Exchange Protocols: Security Definition, Proof Method, and Applications, 19th IEEE Computer Security Foundations Workshop (CSFW 19), Venice, July, 2006.
30. C. Jackson, D. Boneh, J.C. Mitchell, Protecting Browser State from Web Privacy Attacks, 15th International World Wide Web Conference WWW2006, Edinburgh, May, 2006.
31. A. Barth, A. Datta, J. C. Mitchell, H. Nissenbaum, Privacy and Contextual Integrity: Framework and Applications, Proc. 27th IEEE Symposium on Security and Privacy, May 2006.
32. A. Datta, A. Derek, J.C. Mitchell, A. Ramanathan, and A. Scedrov, Games and the Impossibility of Realizable Ideal Functionality, Proceedings Theory of Cryptography Conference (TCC), 2006.
33. C. He, M. Sundararajan, A. Datta, A. Derek, J. C. Mitchell, A Modular Correctness Proof of TLS and IEEE 802.11i, Proc. 12th ACM Conference on Computer and Communications Security, November, 2005. (Invited to ACM Transactions on Information and System Security, Special Issue of Selected Papers from CCS'05.)
34. A. Datta, A. Derek, J.C. Mitchell, V. Shmatikov, and M. Turuani, Probabilistic polynomial-time semantics for a protocol security logic (invited), 32nd International Colloquium on Automata, Languages and Programming (ICALP '05), Lisbon, July, 2005.
35. B. Ross, C. Jackson, N. Miyake, D. Boneh, J.C. Mitchell, Stronger Password Authentication Using Browser Extensions, Usenix Security Symposium, Baltimore, August, 2005.
36. M. Backes, A. Datta, A. Derek, J.C. Mitchell, and M. Turuani, Compositional Analysis of Contract-Signing Protocols, 18th IEEE Computer Security Foundations Workshop (CSFW 18), Aix en Provence, June, 2005.
37. A. Datta, R. Küsters, J. C. Mitchell, A. Ramanathan, On the Relationships between Notions of Simulation-based Security, in Proceedings of Theory of Cryptography Conference, Lecture Notes in Computer Science, Vol. 3378, pp. 476-494, February 2005.
38. C. He and J.C. Mitchell, Security Analysis and Improvements for IEEE 802.11i, 11th Annual Network and Distributed System Security Symposium (NDSS '05), San Diego, February, 2005.
39. A. Barth and J.C. Mitchell, Enterprise privacy promises and enforcement, Workshop on Issues in the Theory of Security (WITS'05), San Diego, January, 2005.

40. N. Li, J.C. Mitchell, and Derrick Tong, Securing Java RMI-based Distributed Applications, 20th Annual Computer Security Applications Conference (ACSAC 2004), Tucson, December, 2004.
41. A. Barth, J.C. Mitchell, and J. Rosenstein, Conflict and Combination in Privacy Policy Languages, ACM Workshop on Privacy in the Electronic Society (WPES), Washington (DC), October, 2004.
42. C. He and J.C. Mitchell, Analysis of the 802.11i 4-Way Handshake, ACM Workshop on Wireless Security (WiSe 2004), Philadelphia, October, 2004.
43. A. Datta and A. Derek and J. C. Mitchell and D. Pavlovic, Abstraction and Refinement in Protocol Derivation, IEEE Computer Security Foundations Workshop, Pacific Grove, California, June 2004.
44. A. Datta, R. Küsters, J.C. Mitchell, A. Ramanathan, V. Shmatikov, Unifying Equivalence-Based Definitions of Protocol Security, Workshop on Issues in the Theory of Security (WITS'04), Barcelona, April, 2004.
45. A. Ramanathan, J.C. Mitchell, A. Scedrov, and V. Teague, Probabilistic bisimulation and equivalence for security analysis of network protocols, Foundations of Software Science and Computation Structures (FOSSACS 2004), Barcelona, March, 2004.
46. N. Chou, R. Ledesma, Y. Teraguchi, and J.C. Mitchell, Client-Side Defense Against Web-Based Identity Theft, 11th Annual Network and Distributed System Security Symposium (NDSS '04), San Diego, February, 2004.
47. A. Datta and A. Derek and J. C. Mitchell and D. Pavlovic, Secure protocol composition, Proc. of Mathematical Foundations of Programming Semantics, Electronic Notes in Theoretical Computer Science, 2003.
48. A. Datta, A. Derek, J. C. Mitchell, D. Pavlovic, Secure protocol composition (Extended abstract), Proc. ACM Workshop on Formal Methods in Security Engineering, 2003, pages 11-23.
49. R. Chadha, J.C. Mitchell, A. Scedrov, and V. Shmatikov, Contract signing, optimism, and advantage, CONCUR 2003, Marseille, France, Springer LNCS Volume 2761, Springer-Verlag, 2003, pp. 366-382.
50. P. Mateus, J.C. Mitchell, and A. Scedrov, Composition of Cryptographic Protocols in a Probabilistic Polynomial-Time Calculus, CONCUR 2003, Marseille, France, Springer LNCS Volume 2761, Springer-Verlag, 2003, pp. 327-349.
51. N. Li and J.C. Mitchell, Understanding SPKI/SDSI Using First-Order Logic, IEEE Computer Security Foundations Workshop, Pacific Grove, California, June 2003, pages 89-103.
52. A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic, A Derivation System for Security Protocols and its Logical Formalization, IEEE Computer Security Foundations Workshop, Pacific Grove, California, June 2003, pages 109-125.

53. N. Li, J.C. Mitchell, W.H. Winsborough, Beyond Proof-of-compliance: Safety and Availability Analysis in Trust Management IEEE Symp. on Security and Privacy, Oakland, May 2003.
54. D. Lie, J. Mitchell, C. Thekkath, M. Horowitz, Specifying and Verifying Hardware for Tamper-Resistant Software, IEEE Symp. on Security and Privacy, Oakland, May 2003.
55. N. Li and J.C. Mitchell, RT: A Role-based Trust-management Framework, DARPA Information Survivability Conference and Exposition (DISCEX III), April, 2003.
56. N. Li and J.C. Mitchell, Datalog with Constraints: A Foundation for Trust-management Languages, Proc. Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003), New Orleans, Louisiana, Springer LNCS Vol. 2562, pp. 58–73 January 2003.
57. J.C. Mitchell and V. Teague, Autonomous Nodes and Distributed Mechanisms, In M. Okada, B. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, eds., *Software Security – Theories and Systems. Next-NSF-JSPS International Symposium, ISSS 2002, Tokyo, Japan, November 8-10, 2002, Revised Papers*, Springer LNCS Volume 2609, Springer-Verlag, 2003, pp. 58–83.
58. I. Cervesato, N. Durgin, J. Mitchell, P. Lincoln, A. Scedrov, A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis, International Symposium on Software Security, In M. Okada, B. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, eds., *Software Security – Theories and Systems. Next-NSF-JSPS International Symposium, ISSS 2002, Tokyo, Japan, November 8-10, 2002, Revised Papers*, Springer LNCS Volume 2609, Springer-Verlag, 2003, pp. 356 - 383.
59. N. Li, J.C. Mitchell, W.H. Winsborough, Design of a Role-based Trust-management Framework, IEEE Symp. on Security and Privacy, Oakland, May 2002.
60. A. Chander, D. Dean, J.C. Mitchell, Deconstructing Trust Management, ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'02) Portland, Oregon, USA, January 14-15, 2002. Electronic proceedings at http://www.dsi.unive.it/IFIPWG1_7/WITS2002/
61. N. Li, W.H. Winsborough, J.C. Mitchell, Distributed Credential Chain Discovery in Trust Management, 8th ACM Computer and Communications Security Conference (CCS 2001), Philadelphia, Nov, 2001.
62. Comon, H., Cortier, V, and Mitchell, J.C., Tree Automata with one Memory, Set Constraints and Ping-Pong Protocols, ICALP 2001, Crete, Greece, July 8-12, 2001.
63. Chander, A., Mitchell, J.C., and Shin, I., Mobile code security by Java bytecode instrumentation, DARPA Information Survivability Conference and Exposition (DISCEX II), June, 2001.
64. Durgin, N.A., Mitchell, J.C., and Pavlovic, D., A Compositional Logic for Protocol Correctness, 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, June 11-13, 2001.

65. Chander, A., Dean, D., and Mitchell, J.C., A state-transition model of trust management and access control, 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, June 11-13, 2001.
66. J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague, A probabilistic polynomial-time calculus for analysis of cryptographic protocols (Preliminary report), 17-th Annual Conference on the Mathematical Foundations of Programming Semantics, Aarhus, Denmark, May, 2001, Electronic Notes in Theoretical Computer Science, Volume 45 (2001).
67. D. Lie, C. Thekkath, P. Lincoln, M. Mitchell, D. Boneh, J. Mitchell, M. Horowitz, Architectural Support for Copy and Tamper Resistant Software, Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), Cambridge, MA, November 12-15, 2000.
68. I. Cervesato, N. Durgin, J. Mitchell, P. Lincoln and A. Scedrov, Relating Strands and Multiset Rewriting for Security Protocol Analysis, 13-th IEEE Computer Security Foundations Workshop, Cambridge, U.K., July 3-5, 2000.
69. Shmatikov, V. and Mitchell, J.C., Analysis of Abuse-Free Contract Signing, Financial Cryptography 00, accepted for publication.
70. Shmatikov, V. and Mitchell, J.C., Analysis of a Fair Exchange Protocol, Seventh Annual Symposium on Network and Distributed System Security (NDSS2000), accepted for publication.
71. Freund, S.N., and Mitchell, J.C., A Formal Framework for the Java Bytecode Language and Verifier, ACM Conference on Object-Oriented Programming: Systems, Languages and Applications, Denver, CO, November, 1999, pages 147-166.
72. Lincoln, P.D., Mitchell, J.C., Mitchell, M., and Scedrov, A., Probabilistic polynomial-time equivalence and security protocols, FM'99 World Congress On Formal Methods in the Development of Computing Systems, Toulouse, France, September, 1999.
73. Durgin, N.A., Lincoln, P.D., Mitchell, J.C., and Scedrov, A., Undecidability of bounded security protocols, Workshop on Formal Methods and Security Protocols (FMSP'99), Trento, Italy, July 5, 1999. Electronic proceedings: <http://www.cs.bell-labs.com/who/nch/fmsp99/program>.
74. Cervesato, I., Durgin, N.A., Lincoln, P.D., Mitchell, J.C., and Scedrov, A., A meta-notation for protocol analysis, 12-th IEEE Computer Security Foundations Workshop, Mordano, Italy, June 28-30, 1999.
75. Bono, V., Patel, A., Shmatikov, V., and Mitchell, J.C., A core calculus of classes and mixins, European Conference on Object-Oriented Programming, 1999 (accepted for publication).
76. Bono, V., Patel, A., Shmatikov, V., and Mitchell, J.C., A core language of classes and objects, 15th Conf. Mathematical Foundations of Programming Semantics, 1999 (accepted for publication).

77. Mitchell, J.C., Mitchell, M., and Scedrov, A., A linguistic characterization of bounded oracle computation and probabilistic polynomial time, IEEE Foundations of Computer Science, Palo Alto, Ca. November 8–11, 1998, pages 725–733.
78. Lincoln, P.D., Mitchell, J.C., Mitchell, M., and Scedrov, A., A probabilistic poly-time framework for protocol analysis, 5th ACM Conference on Computer and Communications Security, San Francisco, Ca., November 3–5, 1998, pages 112–121.
79. Freund, S., and Mitchell, J.C., A type system for object initialization in the Java bytecode language, ACM Symp. Object-Oriented Programming: Systems, Languages and Applications (OOPSLA), Vancouver, October 20–22, 1998, pages 310–328.
80. Mitchell, J.C., Shmatikov, V., and Stern, U., Finite-State Analysis of SSL 3.0, Seventh USENIX Security Symposium, San Antonio, 1998, pages 201–216. Preliminary version presented at DIMACS Workshop on Design and Formal Verification of Security Protocols, September 1997, and distributed on workshop CD.
81. Agesen, O., Freund, S., and Mitchell, J.C., Adding Parameterized Types to Java, ACM Symp. Object-Oriented Programming: Systems, Languages and Applications (OOPSLA), Atlanta, October 7–9, 1997, pages 49–65.
82. Mitchell, J.C., Mitchell, M. and Stern, U., Automated Analysis of Cryptographic Protocols Using Mur ϕ , IEEE Symp. on Security and Privacy, Oakland, May 4–7, 1997, pages 141–151.
83. Lincoln, P.D., Mitchell, J.C. and Scedrov, A., The Complexity of Local Proof Search in Linear Logic (Extended Abstract). In *Proc. Linear Logic '96, Tokyo Meeting*, March 28–April 2, Tokyo, Electronic Notes in Theoretical Computer Science, Volume 3, 1996, 10 pages. <http://www1.elsevier.nl/mcs/tcs/pc/volume3.htm>. (Invited paper.)
84. Lincoln, P.D., Mitchell, J.C. and Scedrov, A., P. Lincoln, J. Mitchell, A. Scedrov "The Complexity Of Local Proof Search In Linear Logic" Workshop on Proof Search in Type Theoretic Languages, in association with CADE 13, New Brunswick, NJ, July 30 - August 3, 1996, pages 69–76. Proc. ed. D. Galmiche.
85. Mitchell, J.C. and Viswanathan, R., Effective models of polymorphism, subtyping and recursion, Proc. 23rd International Colloquium on Automata, Languages, and Programming (ICALP '96), Paderborn, Germany, July 8–12, Springer LNCS 1099, 1996, pages 170–181.
86. Fisher, K. and Mitchell, J.C., A delegation-based object calculus with subtyping, Proc. 10th Int'l Conf. Fundamentals of Computation Theory (FCT'95), Dresden, Germany, August 22–25, Springer LNCS 965, 1995, pages 42–61. (Invited paper.)
87. Hoang, M. and Mitchell, J.C., Lower bounds on type inference with subtypes, Proc. 22nd ACM Symp. on Principles of Programming Languages, San Francisco, CA, January 22–25, 1995, pages 176–185.

88. Fisher, K. and Mitchell, J.C., Notes on typed object-oriented programming, Proc. Theor. Aspects of Computer Software, Sendai, Japan, April 19–22, Springer LNCS 789, 1994, pages 844–885. (Invited paper.)
89. Katiyar, D., Luckham, D., and Mitchell, J.C., A type system for prototyping languages, Proc. 21st ACM Symp. on Principles of Programming Languages, Portland, January 17–21, 1994, pages 138–150.
90. Katiyar, D., Luckham, D., Meldal, S. and Mitchell, J.C., Polymorphism and subtyping in interfaces, ACM Workshop on Interface Definition Languages, 1994. Workshop associated with 21st ACM Symp. on Principles of Programming Languages, Portland, Oregon. Proceedings ed. J. Wing.
91. Mitchell, J.C., Honsell, F. and Fisher, K., A lambda calculus of objects and method specialization, Proc. 8th IEEE Symp. Logic in Computer Science, Montreal, June 19–23, 1993, pages 26–38.
92. Hoang, M., Mitchell, J.C. and Viswanathan, R., Standard ML weak polymorphism and imperative constructs, Proc. 8th IEEE Symp. Logic in Computer Science, Montreal, June 19–23, 1993, pages 15–25.
93. Mitchell, J.C. and Scedrov, A., Notes on scoping and relators, *Computer Science Logic '92, Selected Papers*, E. Börger et al., eds., Springer LNCS 702, 1993, pages 352–378. (Paper fully refereed after conference.)
94. Lincoln, P.D. and Mitchell, J.C., Operational aspects of linear lambda calculus, Proc. 7th IEEE Symp. Logic in Computer Science, Santa Cruz, June 22–25, 1992, pages 235–247.
95. Lincoln, P.D. and Mitchell, J.C., Algorithmic aspects of type inference with subtypes, Proc. 19th ACM Principles of Programming Languages Conf., Albuquerque, January 19–22, 1992, pages 293–304.
96. Bruce, K. and Mitchell, J.C., PER models of subtyping, recursive types and higher-order polymorphism, Proc. 19th ACM Principles of Programming Languages Conf., Albuquerque, January 19–22, 1992, pages 316–327.
97. Kurtz, S.A., Mitchell, J.C. and O'Donnell, M.J., Connecting formal semantics to constructive intuitions, in Myers and O'Donnell, eds., *Constructivity in Computer Science Summer Symposium*, San Antonio, June 19–21, Springer LNCS 613, 1992, pages 1–21.
98. Mitchell, J.C., On abstraction and the expressive power of programming languages, Proc. Theor. Aspects of Computer Software, Sendai, Japan, September 24–27, Springer LNCS 526, 1991, pages 290–310.
99. Cardelli, L., Martini, S., Mitchell, J. and Scedrov, A., An extension of System F with subtyping, Proc. Theor. Aspects of Computer Software, Sendai, Japan, September 24–27, Springer LNCS 526, 1991, pages 750–770.

100. Mitchell, J.C., Meldal, S. and Madhav, N., An extension of Standard ML modules with subtyping and inheritance, Proc. 18th ACM Principles of Programming Languages Conf., Orlando, January 21–23, 1991, pages 270–278.
101. Lincoln, P., Mitchell, J.C., Scedrov, A. and Shankar, N., Decision Problems for Propositional Linear Logic, Proc. 31st IEEE Symp. on Foundations of Computer Science, St. Louis, October 22–24, 1990, pages 662–671.
102. Howard, B. and Mitchell, J.C., Operational and axiomatic semantics of PCF, Proc. ACM Conf. Lisp and Functional Programming, Nice, France, June 27–29, 1990, pages 298–306.
103. Mitchell, J.C., Toward a typed foundation for method specialization and inheritance, Proc. 17th ACM Principles of Programming Languages Conf., San Francisco, January 17–19, 1990, pages 109–124.
104. Harper, R., Mitchell, J.C. and Moggi, E., Higher-order modules and the phase distinction, Proc. 17th ACM Principles of Programming Languages Conf., San Francisco, January 17–19, 1990, pages 341–354.
105. Canning, Cook, Hill, Mitchell and Olthoff, F-Bounded quantification for object-oriented programming, Proc. ACM Conf. Functional Programming and Computer Architecture, London, September 11–13, 1989, pages 273–280.
106. Cardelli, L. and Mitchell, J.C., Operations on records (summary), *Mathematical Foundations of Programming Language Semantics. Proceedings, 1989*. Springer-Verlag LNCS 442, 1990, pages 22–52. (Paper fully refereed after conference.)
107. Kanellakis, P.C. and Mitchell, J.C., Polymorphic unification and ML typing, Proc. 16th ACM Principles of Programming Languages Conf., Austin, January 11–13, 1989, pages 105–115.
108. Jategaonkar, L. and Mitchell, J.C., ML with extended pattern matching and subtypes, ACM Conf. on Lisp and Functional Programming, Snowbird, July 25–27, 1988, pages 198–211.
109. Mitchell, J.C. and Harper, R., The Essence of ML, Proc. 15th ACM Principles of Programming Languages Conf., San Diego, January 13–15, 1988, pages 28–46.
110. Mitchell, J.C. and Scott, P.J., Typed lambda models and cartesian closed categories, *Contemporary Mathematics*, Volume 92, Categories in Computer Science and Logic (Proceedings of a Summer Research Conference held June 14–20, 1987), Amer. Math. Society, 1989, pages 301–316. (Paper fully refereed after conference.)
111. Mitchell, J.C. and Moggi, E., Kripke-style models for typed lambda calculus, Proc. IEEE Symp. on Logic in Computer Science, Ithaca, June 22–25, 1987, pages 303–314.
112. Meyer, A.R., Mitchell, J.C., Moggi, E. and Statman, R., Empty types in polymorphic lambda calculus, Proc. 14th ACM Principles of Programming Languages Conf., Munich, January 21–23, 1987, pages 253–262.

113. Mitchell, J.C., A type inference approach to reduction properties and semantics of polymorphic expressions, Proc. ACM Lisp and Functional Programming Conf., Cambridge, MA, August 4–6, 1986, pages 308–319.
114. Mitchell, J.C. and O'Donnell, M.J., Realizability semantics for error-tolerant logics, Proc. Conf. on Theoretical Aspects of Reasoning About Knowledge, Monterey, CA, March 19–22, 1986, pages 363–382.
115. Mitchell, J.C., Representation independence and data abstraction, Proc. 13th ACM Principles of Programming Languages Conf., St. Petersburg, FL, January 13–15, 1986, pages 263–276.
116. Mitchell, J.C. and Meyer, A.R., Second-order logical relations, Proc. 1985 Logics of Programs, Brooklyn, June 17–19, Springer LNCS 193, 1985, pages 225–237.
117. Mitchell, J.C. and Plotkin, G.D., Abstract types have existential type, Proc. 12th ACM Principles of Programming Languages Conf., New Orleans, January 14–16, 1985, pages 37–51.
118. Mitchell, J.C., Semantic models of second-order lambda calculus, Proc. 25th Annual IEEE Symp. on Foundations of Computer Science, Singer Island, FL, October 24–26, 1984, pages 289–299.
119. Mitchell, J.C., Type inference and type containment. Proc. Int'l Symp. on the Semantics of Data Types, Sophia-Antipolis (France), June 27–29, Springer LNCS 173, 1984, pages 257–278.
120. Mitchell, J.C., Coercion and type inference. Proc. 11th ACM Principles of Programming Languages Conf., Salt Lake City, January 15–18, 1984, pages 175–185.
121. Mitchell, J.C., Inference rules for functional and inclusion dependencies. Proc. Second ACM Symp. on Principles of Database Systems, Atlanta, March 21–23, 1983, pages 58–69.
122. Meyer, A.R. and Mitchell, J.C., Axiomatic definability and completeness for recursive programs, Proc. 9th ACM Principles of Programming Languages Conf., Albuquerque, January 25–27, 1982, pages 337–346.
123. Mitchell, J.C., FCHART 4.0: The University of Wisconsin Solar Energy Design Program. Proc. DOE Systems Simulation and Economic Analysis Conference, January, 1980.

Workshop and Journal Abstracts

1. Freund, S. and Mitchell, J.C., A Type System for Object Initialization in the Java Bytecode Language. Overview of work in progress presented at Second Workshop on Higher-Order Operational Techniques in Semantics, December, 1997. Short summary appears in *Electronic Notes in Theoretical Computer Science* 19 (1998), <http://www.elsevier.nl/locate/entcs/volume10.html>, 4 pages.

2. Mitchell, J.C., Typed lambda calculus and logical relations (abstract), *Journal of Symbolic Logic* (accepted for publication).
3. Mitchell, J.C., Abstract realizability for intuitionistic and relevant implication (abstract), *Journal of Symbolic Logic* 51, 3 (1986), pages 851–852.

Position papers

1. Gunter, C., Mitchell, J.C. and Notkin, D., Strategic Directions in Software Engineering and Programming Languages, *ACM Computing Surveys*, Vol 28A, No 4, December 1996.
2. Harper, R. and Mitchell, J.C., ML and Beyond, *ACM SIGPLAN Notices*, Vol 32, No 1, 1997, pages 80–85. Position statement on strategic directions for research in programming languages, in connection with Strategic Directions in Computing Research report on Programming Languages.

CONFERENCE LECTURES NOT ASSOCIATED WITH PUBLICATION

J. Mitchell, V. Shmatikov, U. Stern, Finite-State Analysis of SSL 3.0 and Related Protocols, DIMACS Workshop on Design and Formal Verification of Security Protocols, New Brunswick, September 3-5, 1997. (Informal publication on web site and CD-ROM.)

Predicative and Impredicative Polymorphism, Spring meeting of the Assoc. for Symbolic Logic, New York City, May, 1987.

Abstract realizability for intuitionistic and relevance logics, Assoc. Symbolic Logic Conf. on Logic, Language and Computation, Stanford CA, July, 1985.

SEMINARS AT UNIVERSITIES AND RESEARCH ORGANIZATIONS

AT&T Bell Labs
 Brown University
 Cambridge University
 Carnegie-Mellon University
 Columbia University
 Cornell University
 CUNY Graduate Center
 Edinburgh University
 IBM San Jose
 IBM Yorktown Heights
 Indian Institute of Technology, Mumbai
 Int'l Computer Science Inst. (Berkeley)
 INRIA Rocquencourt (Paris)
 Keio University (Tokyo)

MIT
Purdue University
Oxford University
Stanford University
SUNY Stony Brook
Tata Research Design and Development Center
Toshiba Corporation (Kawasaki)
University of California, Berkeley
University of Chicago
University of Pennsylvania
Universita di Torino (Italy)
University of Wisconsin
Xerox PARC

REFEREED TUTORIALS

Semantic Methods for Object-Oriented Languages, with Luca Cardelli, 1988 ACM Conference on Object-Oriented Programming: Systems, Languages and Architectures (OOPSLA), 1/2 day, 210 registrants.

PhD Students

1. *Patrick D. Lincoln*: Computational Aspects of Linear Logic. August, 1992. Present position: Director, Computer Science Laboratory, SRI International.
2. *Brian T. Howard*: Fixed Points and Extensionality in Typed Functional Languages. August, 1992. Present position: Bridgewater College.
3. *Dinesh Katiyar*: Theory and Practice of Typed Object-Oriented Programming. December, 1994. Subsequent positions: Software Engineer, Sun Microsystems; Founder & CEO, iLeverage (acquired by Epiphany).
4. *Ramesh Viswanathan*: Recursion Theoretic Semantics, Fully Abstract Term Models, and Imperative Constructs, June, 1995. Present position: Member Technical Staff, Lucent Bell Laboratories.
5. *My Hoang*: Type Inference and Program Evaluation in the Presence of Subtyping, June, 1995. Present position: Software Engineer, SAP International.
6. *Ole Agesen*: Type inference: A path to delivery of dynamically-typed object-oriented applications, December, 1995. (Co-advisee with David Ungar, Sun Microsystems.) Subsequent positions: Member Technical Staff, Sun Microsystems; VMWare, Inc.
7. *Kathleen Fisher*: Type Systems for Object-Oriented Programming, August, 1996. Present position: Member Technical Staff, AT&T Laboratories.

8. *Vitaly Shmatikov*: Finite-State Analysis of Security Protocols, August, 2000. Subsequent positions: Member of Research Staff, Lucent Bell Laboratories; Computer Scientist, SRI International; Assistant Professor, Univ Texas (Austin).
9. *Stephen Freund*: Type Systems for Object-Oriented Intermediate Languages, December, 2000. Subsequent positions: Member of Research Staff, Compaq Systems Research Center; Assistant Professor, Williams College.
10. *Amit Patel*: Obstacle: A language with Objects, Subtyping, and Classes, December, 2001. Present position: Google, Inc.
11. *Mark Mitchell*: Entered Stanford Fall 1996. On leave.
12. *Nancy Durgin*: Logical Analysis and Complexity of Security Protocols, March, 2003. Present position: Senior Member of Technical Staff, Sandia National Laboratories, Livermore, CA.
13. *Ajay Chandar*: A Constructive Design Methodology for Trust Management Systems, October, 2003. Present position: Research Engineer, Mobile Software Laboratory, DoCoMo USA Labs.
14. *Vanessa Teague*: Combining Cryptography and Game Theory in Distributed Algorithms, December 2004. Present position: Lecturer (Assistant Professor), Univ Melbourne.
15. *Ajith Ramanathan*: A Probabilistic Polynomial-time Process Calculus for the Analysis of Cryptographic Protocols, June 2005. Present position: Google, Inc.
16. *Anupam Datta*: Security Analysis of Network Protocols: Compositional Reasoning and Complexity-Theoretic Foundations, September 2005.
17. *Changhua He*: Analysis of Security Protocols for Wireless Networks, December 2005.
18. *Ante Derek*: Formal Analysis of Security Protocols: Protocol Composition Logic, December 2006.
19. *Mukund Sundararajan*: Trade-offs in cost sharing, (principal advisor - Tim Roughgarden), June 2009.
20. *Adam Barth*: Design and Analysis of Privacy Policies, August 2008.
21. *Collin Jackson*: Improving browser security policies, September 2009.
22. *Arnab Roy*: Formal proofs of cryptographic security of network protocols, December 2009.
23. *Elizabeth Stinson*: Entered Stanford PhD Program Fall 2007 (on leave).
24. *Stephan Hyeonjun Stiller*: Network protocol security, PhD expected 2010.
25. *Ankur Taly*: Entered Stanford PhD Program Fall 2007.
26. *Jason Bau*: EE PhD program, PhD expected 2010-11.

27. *Peifung Eric Lam*: Entered Stanford PhD Program Fall 2008.
28. *Jonathan Mayer*: Stanford CS PhD and JD programs, beginning Fall 2009.

S.M. THESIS STUDENT

Lalita Jategaonkar: Supervisor of S.M. thesis research at AT&T Bell Labs under the MIT VI-A Cooperative Program. Thesis completed at MIT, 1989. Title: *ML with Extended Pattern Matching and Subtypes*. MIT supervisor: Albert R. Meyer.

CLASSROOM TEACHING

Stanford University:

Sophomore Seminar: Computer Security and Privacy (CS 99J)

Discrete Mathematics and Logic (CS 103x)

Web Application Development and Security (CS 142)

Intro. to Automata and Complexity Theory (CS 154)

Intro. to Computer Security (CS 155)

Logic and Automated Reasoning (CS 157)

Discrete Structures and Algorithms (CS 161)

Web Security (CS 241)

Programming Languages (CS 242)

Intro. to Programming Language Theory (CS 258)

Security Analysis of Network Protocols (CS 259)

TechLaw with Progressive Minds (CS 302)

Advanced Programming Languages (CS 342)

Topics in Programming Language Theory (CS 358).

CS 154, 157, 161, 258 and 358 are considered “theory” courses;

CS 242 and 342 are considered “systems” courses

CS 142, 155, 241, and 259 are computer security courses developed at Stanford.

New York University: Type Theory and Programming Languages (G22.3033.01), with David MacQueen.

MIT: Teaching Assistant to A. Meyer and J. Stoy, Summer 1982. Lectured on the formal definition of Ada and principles of language design for special summer course on semantics of programming languages.

Teaching Assistant to B. Liskov, Fall 1981. Graduate course 6.821, *Concepts in Modern Programming Languages*.

Exhibit B

List of Materials Considered

1. Complaint for Patent and Copyright Infringement, Demand for Jury Trial, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, August 12, 2010.
2. Google Inc.'s Answer to Plaintiff's Complaint for Patent and Copyright Infringement and Counterclaims, Jury Trial Demanded, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, October 4, 2010.
3. Google Inc.'s Motion to Dismiss Count VIII of Plaintiff's Complaint Or, In The Alternative, For a More Definite Statement, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, October 4, 2010.
4. Amended Complaint for Patent and Copyright Infringement, Demand for Jury Trial, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, October 27, 2010.
5. Defendant Google Inc.'s Supplemental Responses to Plaintiff's Interrogatories, Set One.
6. Plaintiff Oracle America, Inc.'s Supplemental Responses to Defendant Google, Inc.'s Interrogatories, Set No. 1.
7. Plaintiff's Request for Production of Documents and Things to Defendant Google Inc., Set One.
8. Oracle's Technology Tutorial, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, April 6, 2011, Oracle 04-06-2011 Tech Tutorial Slide.PPT.
9. Google's Technical Tutorial, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, April 6, 2011, Oracle v Google – Google Tech Tutorial Slides 04-06-2011.pdf.
10. Oracle's Technology Tutorial, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, April 6, 2011, Oracle 04-06-2011 Tech Tutorial Rebuttal Slide.PPT.
11. Oracle's Technology Tutorial Supplement, *Oracle America, Inc., v. Google, Inc.*, April 6, 2011, Oracle 04-06-2011 Tech Tutorial Supplement.PPT.
12. Claim Construction Hearing, Oracle's Argument, *Oracle America, Inc., v. Google, Inc.*, April 20, 2011, Oracle Markman Hearing Slides 04-20-2011.pdf.

13. Transcript of Proceedings, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, April 20, 2011.
14. Claim Construction Order, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, May 9, 2011.
15. Order Granting in Part Motion to Strike Damage Report of Plaintiff Expert Iain Cockburn, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, July 22, 2011.
16. Order Granting In Part And Denying In Part Plaintiff's Request To Take Additional Depositions, *Oracle America, Inc., v. Google, Inc.*, No. 10-03561 WHA, United States District Court, Northern District of California, July 21, 2011.
17. U.S. Patents Nos. 5,966,702; 6,061,520; 6,125,447; 6,192,476; RE38,104 E; 6,910,205; 7,426,720 and associated file histories.
18. Oracle Java code, documentation, specifications, and related materials.
19. Android code, documentation, specifications, videos, and related materials.
20. Files from Android devices.
21. Deposition transcripts and exhibits: Joshua Bloch, Daniel Bornstein, Dan Morrill, Andrew Rubin, Brian Swetland.
22. Performance benchmarks and testing by Bob Vandette, Noel Poore, and Erez Landau
23. Reports by Johnson-Laird, Inc.
24. Analysis of Samsung Captivate (SGH-I897) & LG (Various Models) Handset Code by Marc Visnick (JLI).
25. Articles/publications:
 - E. Roberts, "The Dream of a Common Language: The Search for Simplicity and Stability in Computer Science Education," SIGCSE '04 Proceedings of the 35th SIGCSE technical symposium on Computer science education (2004), *available at* <http://www-cs-faculty.stanford.edu/~eroberts/papers/SIGCSE-2004/DreamOfACommonLanguage.pdf>
 - M. LaMonica, "IBM, BEA join on Java strategy," CNET News (Nov. 25, 2003), *available at* <http://news.cnet.com/2100-7345-5111567.html>
 - N.C. Schaller et al., Panel Report: Using Java in Computer Science Education, Proceedings of the 2nd Annual Conference on Integrating Technology into

Computer Science Education, ITiCSE 1997, Uppsala, Sweden, 1-5 June, 1997.
 ACM 1997, ISBN 0-89791-923-8, retrieved from
http://portal.acm.org/ft_gateway.cfm?id=266154&type=pdf

- Nielsen Communication Trends, " Highlights from the 2008 Convergence Audit and Consumer Electronics Monitor," Dec. 2008, *available at* http://kr.en.nielsen.com/site/documents/CommunicationTrends_200810.pdf
- Nielsen Communication Trends, "Highlights from the 2009 Nielsen Convergence Audit," *available at* <http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/12/09-Nielsen-Convergence-Audit.pdf>
- NielsenWire, "Who is Winning the U.S. Smartphone Battle?," Mar. 3, 2011, *available at* http://blog.nielsen.com/nielsenwire/online_mobile/who-is-winning-the-u-s-smartphone-battle/
- R. Ierusalimsky, L. H. de Figueiredo, W. Celes, The evolution of Lua, Proceedings of ACM HOPL III (2007) 2-1-2-26, *available at* <http://portal.acm.org/citation.cfm?id=1238846>
- R. Ierusalimsky, L. H. de Figueiredo, W. Celes, The implementation of Lua 5.0, Journal of Universal Computer Science 11 #7 (2005) 1159-1176, *available at* http://www.jucs.org/jucs_11_7/the_implementation_of_lua/jucs_11_7_1159_1176_defigueiredo.pdf
- README from US670(Thunder)_Android_Froyo_USA_USC_Opensource.zip retrieved from <http://www.lg.com/global/support/opensource/opensource.jsp>
- S. Lohr and A. Vance, "I.B.M., Looking to Buy Sun, Sets Up a Software Strategy," The New York Times Inside Technology article (Mar. 18, 2009), *available at* <http://www.nytimes.com/2009/03/19/technology/companies/19sun.html>
- Silicon Graphics International, Standard Template Library Programmer's Guide, http://www.sgi.com/tech/stl/table_of_contents.html
- Stephen G. Kochan, Programming in Objective-C 2.0, Rough Cuts, 2nd Edition, *available at* <http://www.informit.com/articles/article.aspx?p=1271260>
- William R. Cook, Interfaces and Specifications for the Smalltalk-80 Collection Classes, OOPSLA 1992, *available at* <http://portal.acm.org/citation.cfm?id=141938>

26. Internet Websites:

- <http://android.git.kernel.org/>

- <http://blog.nielsen.com/nielsenwire/wp-content/uploads/2010/11/smartphone-OS-share.png>
- <http://code.google.com/webtoolkit/>
- <http://code.google.com/webtoolkit/overview.html>
- <http://developer.android.com>
- <http://developer.android.com/guide/basics/what-is-android.html>
- <http://developer.android.com/guide/practices/compatibility.html>
- <http://developer.android.com/reference/packages.html>
- <http://developer.android.com/sdk/requirements.html>
- <http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/>
- http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/
- <http://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhone101/iPhone101.pdf>
- <http://docs.python.org/tutorial/index.html>
- <http://download.oracle.com/javase/1.5.0/docs/api/>
- <http://download.oracle.com/javase/1.5.0/docs/api/index.html?java/security/ProtectionDomain.html>
- <http://download.oracle.com/javase/1.5.0/docs/api/java/util/Arrays.html>
- <http://download.oracle.com/javase/1.5.0/docs/relnotes/license.html>
- <http://download.oracle.com/javase/tutorial/collections/interfaces/index.html>
- <http://download.oracle.com/javase/tutorial/collections/intro/index.html>
- <http://msdn.microsoft.com/en-us/magazine/cc163855.aspx>
- <http://msdn.microsoft.com/en-us/vcsharp/aa336809>
- <http://obamapacman.com/2011/07/google-engineer-java-alternatives-all-suck-android-needs-to-negotiate-license/>

- <http://patft.uspto.gov/netahtml/PTO/search-bool.html/>
- <http://source.android.com/compatibility/downloads.html>
- <http://source.android.com/source/initializing.html>
- <http://source.android.com/tech/dalvik/dalvik-bytecode.html>
- <http://source.android.com/tech/dalvik/dex-format.html>
- <http://www.businessinsider.com/how-to-use-the-new-android-market-2011-2>
- <http://www.cs.utexas.edu/users/csed/iticse/>
- <http://www.google.com/support/androidmarket/bin/answer.py?hl=en&answer=113407&topic=1100168>
- <http://www.jcp.org>
- <http://www.lua.org/>
- <http://www.microsoft.com/net/overview.aspx>
- <http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html>
- <http://www.oracle.com/technetwork/java/javase/overview/javahistory-timeline-198369.html>
- <http://www.python.org/>
- <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm
- <http://www-01.ibm.com/software/ebusiness/jstart/history.html>
- <https://opensource.samsung.com/>
- www.mozilla.org/js/
- <http://web.archive.org/web/20080205101616/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml>

27. Oracle production:

OAGOOGL0100000454

OAGOOGL0100003560

OAGOOGL0100003653
OAGOOGL0100003796
OAGOOGL0100013203
OAGOOGL0100219511
OAGOOGL0000122862

28. Google production:

GOOGLE-00296158
GOOGLE-00296163
GOOGLE-00296500
GOOGLE-00302662
GOOGLE-00302808
GOOGLE-00320083
GOOGLE-00320116
GOOGLE-00320162
GOOGLE-00381507
GOOGLE-00383073
GOOGLE-00392204
GOOGLE-00392213
GOOGLE-01-00019511
GOOGLE-01-00019527
GOOGLE-01-00019529
GOOGLE-01-00025376
GOOGLE-01-00025576
GOOGLE-01-00029331
GOOGLE-01-00053552
GOOGLE-01-00075935
GOOGLE-02-00111218
GOOGLE-04-00042610
GOOGLE-04-00055098
GOOGLE-04-00055169
GOOGLE-12-00000537
GOOGLE-12-00003871
GOOGLE-12-00006964
GOOGLE-12-00079180
GOOGLE-14-00042244
GOOGLE-22-00171914
GOOGLE-29-00002338

Exhibit Copyright-A

High Level Comparison of Java and Android API Specs

Java™ 2 Platform Standard Edition 5.0 API Specification

Java™ 2 Platform Standard Edition 5.0 API Specification

This document is the API specification for the Java 2 Platform Standard Edition 5.0.

See: [Description](#)

Java 2 Platform Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.

Android APIs

Package Index

These are the Android APIs.

android	Contains the resource classes used by standard Android applications.
android.accessibilityservice	
android.accounts	
android.app	High-level classes encapsulating the overall Android application model.
android.app.admin	
android.app.backup	Contains the backup and restore functionality available to applications. If a user wipes the data on their device or upgrades to a new Android-powered device, all applications that have enabled backup will restore the user's previous data. For a detailed guide to using the backup APIs, see the Data Backup developer guide .

Java™ 2 Platform Standard Edition 5.0 API Specification

The screenshot shows the Java 2 Platform Standard Edition 5.0 API Specification page for the `java.util` package. The page is titled "Package java.util" and contains a description: "Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array)." It also includes a "See:" section with a link to "Description". A table titled "Interface Summary" lists various interfaces and their descriptions:

Interface	Description
<code>Collection<E></code>	The root interface in the <i>collection hierarchy</i> .
<code>Comparator<T></code>	A comparison function, which imposes a <i>total ordering</i> on some collection of objects.
<code>Enumeration<E></code>	An object that implements the <i>Enumeration</i> interface generates a series of elements, one at a time.
<code>EventListener</code>	A tagging interface that all event listener interfaces must extend.
<code>Formatter</code>	The <i>Formatter</i> interface must be implemented by any class that needs to perform custom formatting using the 's' conversion specifier of <i>Formatter</i> .
<code>Iterator<E></code>	An iterator over a collection.
<code>List<E></code>	An ordered collection (also known as a <i>sequence</i>).
<code>ListIterator<E></code>	An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.
<code>Map<K,V></code>	An object that maps keys to values.
<code>Map.Entry<K,V></code>	A map entry (key-value pair).
<code>Observer</code>	A class can implement the <i>Observer</i> interface when it wants to be informed of changes in observable objects.

Android APIs

The screenshot shows the Android Developers website for the `java.util` package. The page is titled "package java.util" and includes a description: "Provides an extensive set of utility classes." It also lists the interfaces and their descriptions:

Interface	Description
<code>Collection<E></code>	<code>Collection</code> is the root of the collection hierarchy.
<code>Comparator<T></code>	A <code>Comparator</code> is used to compare two objects to determine their ordering with respect to each other.
<code>Enumeration<E></code>	A legacy iteration interface.
<code>EventListener</code>	<code>EventListener</code> is the superclass of all event listener interfaces.
<code>Formatter</code>	Classes that handle custom formatting for the 's' specifier of <i>Formatter</i> should implement the <i>Formatter</i> interface.

Java™ 2 Platform Standard Edition 5.0 API Specification

Android APIs

Exhibit Copyright-B

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
Interface Summary		Interfaces	
<u>Appendable</u>	An object to which <code>char</code> sequences and values can be appended.	<u>Appendable</u>	Declares methods to append characters or character sequences.
<u>CharSequence</u>	A <code>CharSequence</code> is a readable sequence of <code>char</code> values.	<u>CharSequence</u>	This interface represents an ordered set of characters and defines the methods to probe them.
<u>Cloneable</u>	A class implements the <code>Cloneable</code> interface to indicate to the <u>Object.clone()</u> method that it is legal for that method to make a field-for-field copy of instances of that class.	<u>Cloneable</u>	This (empty) interface must be implemented by all classes that wish to support cloning.
<u>Comparable<T></u>	This interface imposes a total ordering on the objects of each class that implements it.	<u>Comparable<T></u>	This interface should be implemented by all classes that wish to define a <i>natural order</i> of their instances.
<u>Iterable<T></u>	Implementing this interface allows an object to be the target of the "foreach" statement.	<u>Iterable<T></u>	Instances of classes that implement this interface can be used with the enhanced for loop.
<u>Readable</u>	A <code>Readable</code> is a source of characters.	<u>Readable</u>	Represents a sequence of characters that can be incrementally read (copied) into a <u>CharBuffer</u> .
<u>Runnable</u>	The <code>Runnable</code> interface should be implemented by any class whose instances are intended to be executed by a thread.	<u>Runnable</u>	Represents a command that can be executed.
<u>Thread.UncaughtExceptionHandler</u>	Interface for handlers invoked when a <code>Thread</code> abruptly terminates due to an uncaught exception.	<u>Thread.UncaughtExceptionHandler</u>	Implemented by objects that want to handle cases where a thread is being terminated by an uncaught exception.
Class Summary		Classes	

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
<u>Boolean</u>	The Boolean class wraps a value of the primitive type <code>boolean</code> in an object.	<u>Boolean</u>	The wrapper for the primitive type <code>boolean</code> .
<u>Byte</u>	The Byte class wraps a value of primitive type <code>byte</code> in an object.	<u>Byte</u>	The wrapper for the primitive type <code>byte</code> .
<u>Character</u>	The Character class wraps a value of the primitive type <code>char</code> in an object.	<u>Character</u>	The wrapper for the primitive type <code>char</code> .
<u>Character.Subset</u>	Instances of this class represent particular subsets of the Unicode character set.	<u>Character.Subset</u>	
<u>Character.Unicode Block</u>	A family of character subsets representing the character blocks in the Unicode specification.	<u>Character.Unicode Block</u>	Represents a block of Unicode characters, as defined by the Unicode 4.0.1 specification.
<u>Class<T></u>	Instances of the class <code>Class</code> represent classes and interfaces in a running Java application.	<u>Class<T></u>	The in-memory representation of a Java class.
<u>ClassLoader</u>	A class loader is an object that is responsible for loading classes.	<u>ClassLoader</u>	Loads classes and resources from a repository.
<u>Compiler</u>	The <code>Compiler</code> class is provided to support Java-to-native-code compilers and related services.	<u>Compiler</u>	Placeholder class for environments which explicitly manage the action of a <i>Just In Time (JIT)</i> compiler.
<u>Double</u>	The Double class wraps a value of the primitive type <code>double</code> in an object.	<u>Double</u>	The wrapper for the primitive type <code>double</code> .
<u>Enum<E extends Enum<E>></u>	This is the common base class of all Java language enumeration types.	<u>Enum<E extends Enum<E>></u>	The superclass of all enumerated types.
<u>Float</u>	The Float class wraps a value of primitive type <code>float</code> in an object.	<u>Float</u>	The wrapper for the primitive type <code>float</code> .
<u>InheritableThreadLocal<T></u>	This class extends <code>ThreadLocal</code> to provide inheritance of values from parent thread to child thread: when a child thread is created, the child receives initial values for all inheritable thread-local variables for which the parent has values.	<u>InheritableThreadLocal<T></u>	A thread-local variable whose value is passed from parent to child thread.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
<u>Integer</u>	The Integer class wraps a value of the primitive type <code>int</code> in an object.	<u>integer</u>	The wrapper for the primitive type <code>int</code> .
<u>Long</u>	The Long class wraps a value of the primitive type <code>long</code> in an object.	<u>Long</u>	The wrapper for the primitive type <code>long</code> .
<u>Math</u>	The class <code>Math</code> contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.	<u>Math</u>	Class <code>Math</code> provides basic math constants and operations such as trigonometric functions, hyperbolic functions, exponential, logarithms, etc.
<u>Number</u>	The abstract class <code>Number</code> is the superclass of classes <code>BigDecimal</code> , <code>BigInteger</code> , <code>Byte</code> , <code>Double</code> , <code>Float</code> , <code>Integer</code> , <code>Long</code> , and <code>Short</code> .	<u>Number</u>	The abstract superclass of the classes which represent numeric base types (that is <code>Byte</code> , <code>Short</code> , <code>Integer</code> , <code>Long</code> , <code>Float</code> , and <code>Double</code>).
<u>Object</u>	Class <code>Object</code> is the root of the class hierarchy.	<u>Object</u>	The root class of the Java class hierarchy.
<u>Package</u>	<code>Package</code> objects contain version information about the implementation and specification of a Java package.	<u>Package</u>	Contains information about a Java package.
<u>Process</u>	The <u>ProcessBuilder.start()</u> and <u>Runtime.exec</u> methods create a native process and return an instance of a subclass of <code>Process</code> that can be used to control the process and obtain information about it.	<u>Process</u>	Represents an external process.
<u>ProcessBuilder</u>	This class is used to create operating system processes.	<u>ProcessBuilder</u>	Creates operating system processes.
<u>Runtime</u>	Every Java application has a single instance of class <code>Runtime</code> that allows the application to interface with the environment in which the application is running.	<u>Runtime</u>	Allows Java applications to interface with the environment in which they are running.
<u>RuntimePermission</u>	This class is for runtime permissions.	<u>RuntimePermission</u>	Represents the permission to execute a runtime-related function.
<u>SecurityManager</u>	The security manager is a class that allows		

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	applications to implement a security policy.	SecurityManager	Warning: security managers do not provide a secure environment for executing untrusted code.
Short	The <code>Short</code> class wraps a value of primitive type <code>short</code> in an object.	Short	The wrapper for the primitive type <code>short</code> .
StackTraceElement	An element in a stack trace, as returned by Throwable.getStackTrace() .	StackTraceElement	A representation of a single stack frame.
StrictMath	The class <code>StrictMath</code> contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.	StrictMath	Class <code>StrictMath</code> provides basic math constants and operations such as trigonometric functions, hyperbolic functions, exponential, logarithms, etc.
String	The <code>String</code> class represents character strings.	String	An immutable sequence of characters/code units (<code>chars</code>).
StringBuffer	A thread-safe, mutable sequence of characters.	StringBuffer	A modifiable sequence of characters for use in creating strings, where all accesses are synchronized.
StringBuilder	A mutable sequence of characters.	StringBuilder	A modifiable sequence of characters for use in creating strings.
System	The <code>System</code> class contains several useful class fields and methods.	System	Provides access to system-related information and resources including standard input and output.
Thread	A <i>thread</i> is a thread of execution in a program.	Thread	A Thread is a concurrent unit of execution.
ThreadGroup	A thread group represents a set of threads.	ThreadGroup	ThreadGroup is a means of organizing threads into a hierarchical structure.
ThreadLocal<T>	This class provides thread-local variables.	ThreadLocal<T>	implements a thread-local storage, that is, a variable for which each thread has its own value.
Throwable	The <code>Throwable</code> class is the superclass of all errors and exceptions in the Java language.	Throwable	The superclass of all classes which can be thrown by the virtual machine.
Void	The <code>Void</code> class is an uninstantiable placeholder		

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	class to hold a reference to the Class object representing the Java keyword void.	Void	Placeholder class for the Java keyword <code>void</code> .
Enum Summary		Enums	
<u>Thread.State</u>	A thread state.	<u>Thread.State</u>	A representation of a thread's state.
Exception Summary		Exceptions	
<u>ArithmeticException</u>	Thrown when an exceptional arithmetic condition has occurred.	<u>ArithmeticException</u>	Thrown when the an invalid arithmetic operation is attempted.
<u>ArrayIndexOutOfBoundsException</u>	Thrown to indicate that an array has been accessed with an illegal index.	<u>ArrayIndexOutOfBoundsException</u>	Thrown when the an array is indexed with a value less than zero, or greater than or equal to the size of the array.
<u>ArrayStoreException</u>	Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.	<u>ArrayStoreException</u>	Thrown when a program attempts to store an element of an incompatible type in an array.
<u>ClassCastException</u>	Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.	<u>ClassCastException</u>	Thrown when a program attempts to cast a an object to a type with which it is not compatible.
<u>ClassNotFoundException</u>	Thrown when an application tries to load in a class through its string name using: The <code>forName</code> method in class <code>Class</code> .	<u>ClassNotFoundException</u>	Thrown when a class loader is unable to find a class.
<u>CloneNotSupportedException</u>	Thrown to indicate that the <code>clone</code> method in class <code>Object</code> has been called to clone an object, but that the object's class does not implement the <code>Cloneable</code> interface.	<u>CloneNotSupportedException</u>	Thrown when a program attempts to clone an object which does not support the <code>Cloneable</code> interface.
<u>EnumConstantNotPresentException</u>	Thrown when an application tries to access an enum constant by name and the enum type contains no constant with the specified name.	<u>EnumConstantNotPresentException</u>	Thrown if an <code>enum</code> constant does not exist for a particular name.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
<u>Exception</u>	The class <code>Exception</code> and its subclasses are a form of <code>Throwable</code> that indicates conditions that a reasonable application might want to catch.	<u>Exception</u>	<code>Exception</code> is the superclass of all classes that represent recoverable exceptions.
<u>IllegalAccessException</u>	An <code>IllegalAccessException</code> is thrown when an application tries to reflectively create an instance (other than an array), set or get a field, or invoke a method, but the currently executing method does not have access to the definition of the specified class, field, method or constructor.	<u>IllegalAccess Exception</u>	Thrown when a program attempts to access a field or method which is not accessible from the location where the reference is made.
<u>IllegalArgumentEx ception</u>	Thrown to indicate that a method has been passed an illegal or inappropriate argument.	<u>IllegalArgument Exception</u>	Thrown when a method is invoked with an argument which it can not reasonably deal with.
<u>IllegalMonitor StateException</u>	Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.	<u>IllegalMonitorState Exception</u>	Thrown when a monitor operation is attempted when the monitor is not in the correct state, for example when a thread attempts to exit a monitor which it does not own.
<u>IllegalState Exception</u>	Signals that a method has been invoked at an illegal or inappropriate time.	<u>IllegalState Exception</u>	Thrown when an action is attempted at a time when the virtual machine is not in the correct state.
<u>IllegalThread StateException</u>	Thrown to indicate that a thread is not in an appropriate state for the requested operation.	<u>IllegalThreadState Exception</u>	Thrown when an operation is attempted which is not possible given the state that the executing thread is in.
<u>IndexOutOf BoundsException</u>	Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.	<u>IndexOutOfBounds Exception</u>	Thrown when a program attempts to access a value in an indexable collection using a value which is outside of the range of valid indices.
<u>Instantiation Exception</u>	Thrown when an application tries to create an instance of a class using the <code>newInstance</code> method in class <code>Class</code> , but the specified class object cannot be instantiated because it is an	<u>Instantiation Exception</u>	Thrown when a program attempts to access a constructor which is not accessible from the location where the reference is made.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	interface or is an abstract class.		
<u>InterruptedException</u>	Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the <code>interrupt</code> method in class <code>Thread</code> .	<u>InterruptedException</u>	Thrown when a waiting thread is activated before the condition it was waiting for has been satisfied.
<u>NegativeArraySizeException</u>	Thrown if an application tries to create an array with negative size.	<u>NegativeArraySizeException</u>	Thrown when an attempt is made to create an array with a size of less than zero.
<u>NoSuchFieldException</u>	Signals that the class doesn't have a field of a specified name.	<u>NoSuchFieldException</u>	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a field that does not exist.
<u>NoSuchMethodException</u>	Thrown when a particular method cannot be found.	<u>NoSuchMethodException</u>	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a method that does not exist.
<u>NullPointerException</u>	Thrown when an application attempts to use <code>null</code> in a case where an object is required.	<u>NullPointerException</u>	Thrown when a program tries to access a field or method of an object or an element of an array when there is no instance or array to use, that is if the object or array points to <code>null</code> .
<u>NumberFormatException</u>	Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.	<u>NumberFormatException</u>	Thrown when an invalid value is passed to a string-to-number conversion method.
<u>RuntimeException</u>	<code>RuntimeException</code> is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.	<u>RuntimeException</u>	<code>RuntimeException</code> is the superclass of all classes that represent exceptional conditions which occur as a result of executing an application in the virtual machine.
<u>SecurityException</u>	Thrown by the security manager to indicate a security violation.	<u>SecurityException</u>	Thrown when a security manager check fails.
<u>StringIndexOutOfBoundsException</u>	Thrown by <code>String</code> methods to indicate that an index is either negative or greater than the size	<u>StringIndexOutOfBoundsException</u>	Thrown when the a string is indexed with a value less than zero, or greater than or equal to the size of the

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)																									
Exception	of the string.	BoundsException	array.																								
TypeNotPresentException	Thrown when an application tries to access a type using a string representing the type's name, but no definition for the type with the specified name can be found.	TypeNotPresentException	Thrown when a program tries to access a class, interface, enum or annotation type through a string that contains the type's name and the type cannot be found.																								
UnsupportedOperationException	Thrown to indicate that the requested operation is not supported.	UnsupportedOperationException	Thrown when an unsupported operation is attempted.																								
<div>Error Summary</div> <table><tr><td>AbstractMethodError</td><td>Thrown when an application tries to call an abstract method.</td></tr><tr><td>AssertionError</td><td>Thrown to indicate that an assertion has failed.</td></tr><tr><td>ClassCircularityError</td><td>Thrown when a circularity has been detected while initializing a class.</td></tr><tr><td>ClassFormatError</td><td>Thrown when the Java Virtual Machine attempts to read a class file and determines that the file is malformed or otherwise cannot be interpreted as a class file.</td></tr><tr><td>Error</td><td>An <code>Error</code> is a subclass of <code>Throwable</code> that indicates serious problems that a reasonable application should not try to catch.</td></tr><tr><td>ExceptionInInitializerError</td><td>Signals that an unexpected exception has occurred in a static initializer.</td></tr></table>		AbstractMethodError	Thrown when an application tries to call an abstract method.	AssertionError	Thrown to indicate that an assertion has failed.	ClassCircularityError	Thrown when a circularity has been detected while initializing a class.	ClassFormatError	Thrown when the Java Virtual Machine attempts to read a class file and determines that the file is malformed or otherwise cannot be interpreted as a class file.	Error	An <code>Error</code> is a subclass of <code>Throwable</code> that indicates serious problems that a reasonable application should not try to catch.	ExceptionInInitializerError	Signals that an unexpected exception has occurred in a static initializer.	<div>Errors</div> <table><tr><td>AbstractMethodError</td><td>Thrown by the virtual machine when an abstract method is called.</td></tr><tr><td>AssertionError</td><td>Thrown when an assertion has failed.</td></tr><tr><td>ClassCircularityError</td><td>Thrown when the virtual machine notices that an attempt is made to load a class which would directly or indirectly inherit from one of its subclasses.</td></tr><tr><td>ClassFormatError</td><td>Thrown by a class loader when a class file has an illegal format or if the data that it contains can not be nterpreted as a class.</td></tr><tr><td>Error</td><td><code>Error</code> is the superclass of all classes that represent unrecoverable errors.</td></tr><tr><td>ExceptionInInitializerError</td><td>Thrown when an exception occurs during class nitialization.</td></tr></table>		AbstractMethodError	Thrown by the virtual machine when an abstract method is called.	AssertionError	Thrown when an assertion has failed.	ClassCircularityError	Thrown when the virtual machine notices that an attempt is made to load a class which would directly or indirectly inherit from one of its subclasses.	ClassFormatError	Thrown by a class loader when a class file has an illegal format or if the data that it contains can not be nterpreted as a class.	Error	<code>Error</code> is the superclass of all classes that represent unrecoverable errors.	ExceptionInInitializerError	Thrown when an exception occurs during class nitialization.
AbstractMethodError	Thrown when an application tries to call an abstract method.																										
AssertionError	Thrown to indicate that an assertion has failed.																										
ClassCircularityError	Thrown when a circularity has been detected while initializing a class.																										
ClassFormatError	Thrown when the Java Virtual Machine attempts to read a class file and determines that the file is malformed or otherwise cannot be interpreted as a class file.																										
Error	An <code>Error</code> is a subclass of <code>Throwable</code> that indicates serious problems that a reasonable application should not try to catch.																										
ExceptionInInitializerError	Signals that an unexpected exception has occurred in a static initializer.																										
AbstractMethodError	Thrown by the virtual machine when an abstract method is called.																										
AssertionError	Thrown when an assertion has failed.																										
ClassCircularityError	Thrown when the virtual machine notices that an attempt is made to load a class which would directly or indirectly inherit from one of its subclasses.																										
ClassFormatError	Thrown by a class loader when a class file has an illegal format or if the data that it contains can not be nterpreted as a class.																										
Error	<code>Error</code> is the superclass of all classes that represent unrecoverable errors.																										
ExceptionInInitializerError	Thrown when an exception occurs during class nitialization.																										

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
<u>IllegalAccessError</u>	Thrown if an application attempts to access or modify a field, or to call a method that it does not have access to.	<u>IllegalAccessError</u>	Thrown when the virtual machine notices that a program tries access a field which is not accessible from where it is referenced.
<u>IncompatibleClassChangeError</u>	Thrown when an incompatible class change has occurred to some class definition.	<u>IncompatibleClassChangeError</u>	<u>IncompatibleClassChangeError</u> is the superclass of all classes which represent errors that occur when inconsistent class files are loaded into the same running image.
<u>InstantiationError</u>	Thrown when an application tries to use the Java <code>new</code> construct to instantiate an abstract class or an interface.	<u>InstantiationError</u>	Thrown when the virtual machine notices that a program tries to create a new instance of a class which has no visible constructors from the location where <code>new</code> is invoked.
<u>InternalError</u>	Thrown to indicate some unexpected internal error has occurred in the Java Virtual Machine.	<u>InternalError</u>	Thrown when the virtual machine notices that it has gotten into an undefined state.
<u>LinkageError</u>	Subclasses of <code>LinkageError</code> indicate that a class has some dependency on another class; however, the latter class has incompatibly changed after the compilation of the former class.	<u>LinkageError</u>	<u>LinkageError</u> is the superclass of all error classes that occur when loading and linking class files.
<u>NoClassDefFoundError</u>	Thrown if the Java Virtual Machine or a <code>ClassLoader</code> instance tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the <code>new</code> expression) and no definition of the class could be found.	<u>NoClassDefFoundError</u>	Thrown when the virtual machine is unable to locate a class which it has been asked to load.
<u>NoSuchFieldError</u>	Thrown if an application tries to access or modify a specified field of an object, and that object no longer has that field.	<u>NoSuchFieldError</u>	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a field that does not exist.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
<u>NoSuchMethodError</u>	Thrown if an application tries to call a specified method of a class (either static or instance), and that class no longer has a definition of that method.	<u>NoSuchMethodError</u>	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a method that does not exist.
<u>OutOfMemoryError</u>	Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.	<u>OutOfMemoryError</u>	Thrown when a request for memory is made that can not be satisfied using the available platform resources.
<u>StackOverflowError</u>	Thrown when a stack overflow occurs because an application recurses too deeply.	<u>StackOverflowError</u>	Thrown when the depth of the callstack of the running program exceeds some platform or virtual machine specific limit.
<u>ThreadDeath</u>	An instance of <code>ThreadDeath</code> is thrown in the victim thread when the <code>stop</code> method with zero arguments in class <code>Thread</code> is called.	<u>ThreadDeath</u>	<code>ThreadDeath</code> is thrown when a thread stops executing.
<u>UnknownError</u>	Thrown when an unknown but serious exception has occurred in the Java Virtual Machine.	<u>UnknownError</u>	Thrown when the virtual machine must throw an error which does not match any known exceptional condition.
<u>UnsatisfiedLinkError</u>	Thrown if the Java Virtual Machine cannot find an appropriate native-language definition of a method declared <code>native</code> .	<u>UnsatisfiedLinkError</u>	Thrown when an attempt is made to invoke a native for which an implementation could not be found.
<u>UnsupportedClassVersionError</u>	Thrown when the Java Virtual Machine attempts to read a class file and determines that the major and minor version numbers in the file are not supported.	<u>UnsupportedClassVersionError</u>	Thrown when an attempt is made to load a class with a format version that is not supported by the virtual machine.
<u>VerifyError</u>	Thrown when the "verifier" detects that a class file, though well formed, contains some sort of internal inconsistency or security problem.	<u>VerifyError</u>	Thrown when the virtual machine notices that an attempt is made to load a class which does not pass the class verification phase.
<u>VirtualMachineError</u>	Thrown to indicate that the Java Virtual Machine is broken or has run out of resources	<u>VirtualMachineError</u>	<u>VirtualMachineError</u> is the superclass of all error classes that occur during the operation of the virtual machine.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)
	necessary for it to continue operating.	
Annotation Types Summary		
<u>Deprecated</u>	A program element annotated @Deprecated is one that programmers are discouraged from using, typically because it is dangerous, or because a better alternative exists.	
<u>Override</u>	Indicates that a method declaration is intended to override a method declaration in a superclass.	
<u>Suppress Warnings</u>	Indicates that the named compiler warnings should be suppressed in the annotated element (and in all program elements contained in the annotated element).	

Exhibit Copyright-C

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
Interface Summary		Interfaces	
<u>Closeable</u>	A <code>Closeable</code> is a source or destination of data that can be closed.	<u>Closeable</u>	Defines an interface for classes that can (or need to) be closed once they are not used any longer.
<u>DataInput</u>	The <code>DataInput</code> interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.	<u>DataInput</u>	Defines an interface for classes that are able to read typed data from some source.
<u>DataOutput</u>	The <code>DataOutput</code> interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.	<u>DataOutput</u>	Defines an interface for classes that are able to write typed data to some target.
<u>Externalizable</u>	Only the identity of the class of an <code>Externalizable</code> instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances.	<u>Externalizable</u>	Defines an interface for classes that want to be serializable, but have their own binary representation.
<u>FileFilter</u>	A filter for abstract pathnames.	<u>FileFilter</u>	An interface for filtering <code>File</code> objects based on their names or other information.
<u>FilenameFilter</u>	Instances of classes that implement this interface are used to filter filenames.	<u>FilenameFilter</u>	An interface for filtering <code>File</code> objects based on their names or the directory they reside in.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
<u>Flushable</u>	A <code>Flushable</code> is a destination of data that can be flushed.	<u>Flushable</u>	Defines an interface for classes that can (or need to) be flushed, typically before some output processing is considered to be finished and the object gets closed.
<u>ObjectInput</u>	<code>ObjectInput</code> extends the <code>DataInput</code> interface to include the reading of objects.	<u>ObjectInput</u>	Defines an interface for classes that allow reading serialized objects.
<u>ObjectInputValidation</u>	Callback interface to allow validation of objects within a graph.	<u>ObjectInputValidation</u>	A callback interface for post-deserialization checks on objects.
<u>ObjectOutput</u>	<code>ObjectOutput</code> extends the <code>DataOutput</code> interface to include writing of objects.	<u>ObjectOutput</u>	Defines an interface for classes that allow reading serialized objects.
<u>ObjectStreamConstants</u>	Constants written into the Object Serialization Stream.	<u>ObjectStreamConstants</u>	A helper interface with constants used by the serialization implementation.
<u>Serializable</u>	Serializability of a class is enabled by the class implementing the <code>java.io.Serializable</code> interface.	<u>Serializable</u>	An empty marker interface for classes that want to support serialization and deserialization based on the <code>ObjectOutputStream</code> and <code>ObjectInputStream</code> classes.
Class Summary		Classes	
<u>BufferedInputStream</u>	A <code>BufferedInputStream</code> adds functionality to another input stream—namely, the ability to buffer the input and to support the <code>mark</code> and <code>reset</code> methods.	<u>BufferedInputStream</u>	Wraps an existing <code>InputStream</code> and <i>buffers</i> the input.
<u>BufferedOutputStream</u>	The class implements a buffered output stream.	<u>BufferedOutputStream</u>	Wraps an existing <code>OutputStream</code> and <i>buffers</i> the output.
<u>BufferedReader</u>	Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays,	<u>BufferedReader</u>	Wraps an existing <code>Reader</code> and <i>buffers</i> the input.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
	and lines.	BufferedWriter	Wraps an existing Writer and <i>buffers</i> the output.
BufferedWriter	Write text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.		
ByteArrayInputStream	A <code>ByteArrayInputStream</code> contains an internal buffer that contains bytes that may be read from the stream.	ByteArrayInputStream	A specialized InputStream for reading the contents of a byte array.
ByteArrayOutputStream	This class implements an output stream in which the data is written into a byte array.	ByteArrayOutputStream	A specialized OutputStream for class for writing content to an (internal) byte array.
CharArrayReader	This class implements a character buffer that can be used as a character-input stream.	CharArrayReader	A specialized Reader for reading the contents of a char array.
CharArrayWriter	This class implements a character buffer that can be used as an <code>Writer</code> .	CharArrayWriter	A specialized Writer for class for writing content to an (internal) char array.
DataInputStream	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.	Console	Provides access to the console, if available.
DataOutputStream	A data output stream lets an application write primitive Java data types to an output stream in a portable way.	DataInputStream	Wraps an existing InputStream and reads typed data from it.
File	An abstract representation of file and directory pathnames.	DataOutputStream	Wraps an existing OutputStream and writes typed data to it.
FileDescriptor	Instances of the file descriptor class serve	File	An "abstract" representation of a file system entity identified by a pathname.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
	as an opaque handle to the underlying machine-specific structure representing an open file, an open socket, or another source or sink of bytes.	FileDescriptor	The lowest-level representation of a file, device, or socket.
FileInputStream	A <code>FileInputStream</code> obtains input bytes from a file in a file system.	FileInputStream	A specialized InputStream that reads from a file in the file system.
FileOutputStream	A file output stream is an output stream for writing data to a <code>File</code> or to a <code>FileDescriptor</code> .	FileOutputStream	A specialized OutputStream that writes to a file in the file system.
FilePermission	This class represents access to a file or directory.	FilePermission	A permission for accessing a file or directory.
FileReader	Convenience class for reading character files.	FileReader	A specialized Reader that reads from a file in the file system.
FileWriter	Convenience class for writing character files.	FileWriter	A specialized Writer that writes to a file in the file system.
FilterInputStream	A <code>FilterInputStream</code> contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.	FilterInputStream	Wraps an existing InputStream and performs some transformation on the input data while it is being read.
FilterOutputStream	This class is the superclass of all classes that filter output streams.	FilterOutputStream	Wraps an existing OutputStream and performs some transformation on the output data while it is being written.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
<u>FilterReader</u>	Abstract class for reading filtered character streams.	<u>FilterReader</u>	Wraps an existing <u>Reader</u> and performs some transformation on the input data while it is being read.
<u>FilterWriter</u>	Abstract class for writing filtered character streams.	<u>FilterWriter</u>	Wraps an existing <u>Writer</u> and performs some transformation on the output data while it is being written.
<u>InputStream</u>	This abstract class is the superclass of all classes representing an input stream of bytes.	<u>nputStream</u>	The base class for all input streams.
<u>InputStreamReader</u>	An <u>InputStreamReader</u> is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified <u>charset</u> .	<u>nputStreamReader</u>	A class for turning a byte stream into a character stream.
<u>LineNumberInputStream</u>	Deprecated. <i>This class incorrectly assumes that bytes adequately represent characters.</i>	<u>LineNumberInputStream</u>	<i>This class is deprecated. Use <u>LineNumberReader</u></i>
<u>LineNumberReader</u>	A buffered character-input stream that keeps track of line numbers.	<u>LineNumberReader</u>	Wraps an existing <u>Reader</u> and counts the line terminators encountered while reading the data.
<u>ObjectInputStream</u>	An <u>ObjectInputStream</u> deserializes primitive data and objects previously written using an <u>ObjectOutputStream</u> .	<u>ObjectInputStream</u>	A specialized <u>InputStream</u> that is able to read (deserialize) Java objects as well as primitive data types (int, byte, char etc.).
<u>ObjectInputStream.GetField</u>	Provide access to the persistent fields read from the input stream.	<u>ObjectInputStream.GetField</u>	<u>GetField</u> is an inner class that provides access to the persistent fields read from the source stream.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
<u>ObjectOutputStream</u>	An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.	<u>ObjectOutputStream</u>	A specialized <u>OutputStream</u> that is able to write (serialize) Java objects as well as primitive data types (int, byte, char etc.).
<u>ObjectOutputStream.PutField</u>	Provide programmatic access to the persistent fields to be written to ObjectOutputStream.	<u>ObjectOutputStream.PutField</u>	PutField is an inner class to provide access to the persistent fields that are written to the target stream.
<u>ObjectStreamClass</u>	Serialization's descriptor for classes.	<u>ObjectStreamClass</u>	Represents a descriptor for identifying a class during serialization and deserialization.
<u>ObjectStreamField</u>	A description of a Serializable field from a Serializable class.	<u>ObjectStreamField</u>	Describes a field for the purpose of serialization.
<u>OutputStream</u>	This abstract class is the superclass of all classes representing an output stream of bytes.	<u>OutputStream</u>	The base class for all output streams.
<u>OutputStreamWriter</u>	An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are encoded into bytes using a specified <u>charset</u> .	<u>OutputStreamWriter</u>	A class for turning a character stream into a byte stream.
<u>PipedInputStream</u>	A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream.	<u>PipedInputStream</u>	Receives information from a communications pipe.
<u>PipedOutputStream</u>	A piped output stream can be connected to a piped input stream to create a communications pipe.	<u>PipedOutputStream</u>	Places information on a communications pipe.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
<u>PipedReader</u>	Piped character-input streams.	<u>PipedReader</u>	Receives information on a communications pipe.
<u>PipedWriter</u>	Piped character-output streams.	<u>PipedWriter</u>	Places information on a communications pipe.
<u>PrintStream</u>	A <code>PrintStream</code> adds functionality to another output stream, namely the ability to print representations of various data values conveniently.	<u>PrintStream</u>	Wraps an existing <u>OutputStream</u> and provides convenience methods for writing common data types in a human readable format.
<u>PrintWriter</u>	Print formatted representations of objects to a text-output stream.	<u>PrintWriter</u>	Wraps either an existing <u>OutputStream</u> or an existing <u>Writer</u> and provides convenience methods for printing common data types in a human readable format.
<u>PushbackInputStream</u>	A <code>PushbackInputStream</code> adds functionality to another input stream, namely the ability to "push back" or "unread" one byte.	<u>PushbackInputStream</u>	Wraps an existing <u>InputStream</u> and adds functionality to "push back" bytes that have been read, so that they can be read again.
<u>PushbackReader</u>	A character-stream reader that allows characters to be pushed back into the stream.	<u>PushbackReader</u>	Wraps an existing <u>Reader</u> and adds functionality to "push back" characters that have been read, so that they can be

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
<u>RandomAccessFile</u>	Instances of this class support both reading and writing to a random access file.		read again.
<u>Reader</u>	Abstract class for reading character streams.	<u>RandomAccessFile</u>	Allows reading from and writing to a file in a random-access manner.
<u>SequenceInputStream</u>	A <code>SequenceInputStream</code> represents the logical concatenation of other input streams.	<u>Reader</u>	The base class for all readers.
<u>SerializablePermission</u>	This class is for Serializable permissions.	<u>SequenceInputStream</u>	Concatenates two or more existing <code>InputStreamS</code> .
<u>StreamTokenizer</u>	The <code>StreamTokenizer</code> class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time.	<u>SerializablePermission</u>	Is used to enable access to potentially unsafe serialization operations.
<u>StringBufferInputStream</u>	Deprecated. <i>This class does not properly convert characters into bytes.</i>	<u>StreamTokenizer</u>	Parses a stream into a set of defined tokens, one at a time.
<u>StringReader</u>	A character stream whose source is a string.	<u>StringBufferInputStream</u>	<i>This class is deprecated. Use <code>StringReader</code></i>
<u>StringWriter</u>	A character stream that collects its output in a string buffer, which can then be used to construct a string.	<u>StringReader</u>	A specialized <code>Reader</code> that reads characters from a <code>String</code> in a sequential manner.
<u>Writer</u>	Abstract class for writing to character streams.	<u>StringWriter</u>	A specialized <code>Writer</code> that writes characters to a <code>StringBuffer</code> in a

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
			sequential manner, appending them in the process.
		Writer	The base class for all writers.
Exception Summary		Exceptions	
CharConversionException	Base class for character conversion exceptions.	CharConversionException	The top level class for character conversion exceptions.
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input.	EOFException	Thrown when a program encounters the end of a file or stream during an input operation.
FileNotFoundException	Signals that an attempt to open the file denoted by a specified pathname has failed.	FileNotFoundException	Thrown when a file specified by a program cannot be found.
InterruptedIOException	Signals that an I/O operation has been interrupted.	InterruptedIOException	Signals that a blocking I/O operation has been interrupted.
InvalidClassException	Thrown when the Serialization runtime detects one of the following problems with a Class.	InvalidClassException	Signals a problem during the serialization or or deserialization of an object.
InvalidObjectException	Indicates that one or more deserialized objects failed validation tests.	InvalidObjectException	Signals that, during deserialization, the validation of an object has failed.
IOException	Signals that an I/O exception of some sort has occurred.		

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
<u>NotActiveException</u>	Thrown when serialization or deserialization is not active.	<u>OException</u>	Signals a general, I/O-related error.
<u>NotSerializableException</u>	Thrown when an instance is required to have a Serializable interface.	<u>NotActiveException</u>	Signals that a serialization-related method has been invoked in the wrong place.
<u>ObjectStreamException</u>	Superclass of all exceptions specific to Object Stream classes.	<u>NotSerializableException</u>	Signals that an object that is not serializable has been passed into the <code>ObjectOutput.writeObject()</code> method.
<u>OptionalDataException</u>	Exception indicating the failure of an object read operation due to unread primitive data, or the end of data belonging to a serialized object in the stream.	<u>ObjectStreamException</u>	Signals some sort of problem during either serialization or deserialization of objects.
<u>StreamCorruptedException</u>	Thrown when control information that was read from an object stream violates internal consistency checks.	<u>OptionalDataException</u>	Signals that the <code>ObjectInputStream</code> class encountered a primitive type (<code>int</code> , <code>char</code> etc.) instead of an object instance in the input stream.
<u>SyncFailedException</u>	Signals that a sync operation has failed.	<u>StreamCorruptedException</u>	Signals that the <code>readObject()</code> method could not read an object due to missing information (for example, a cyclic reference that doesn't match a previous instance, or a missing class descriptor for the object to be loaded).
<u>UnsupportedEncodingException</u>	The Character Encoding is not supported.	<u>SyncFailedException</u>	Signals that the <code>sync()</code> method has failed to complete.
<u>UTFDataFormatException</u>	Signals that a malformed string in <u>modified UTF-8</u> format has been read in a data input stream or by any class	<u>UnsupportedEncodingException</u>	Thrown when a program asks for a particular character converter that is unavailable.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
	that implements the data input interface.	UTFDataFormatException	Signals that an incorrectly encoded UTF-8 string has been encountered, most likely while reading some DataInputStream .
WriteAbortedException	Signals that one of the <code>ObjectStreamExceptions</code> was thrown during a write operation.	WriteAbortedException	Signals that the <code>readObject()</code> method has detected an exception marker in the input stream.

Exhibit Copyright-D

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Interface Summary		Interfaces	
<u>Certificate</u>	Deprecated. A new certificate handling package is created in the Java 2 platform.	<u>Certificate</u>	<i>This interface is deprecated. Replaced by behavior in <u>java.security.cert</u></i>
<u>DomainCombiner</u>	A DomainCombiner provides a means to dynamically update the ProtectionDomains associated with the current AccessControlContext.	<u>DomainCombiner</u>	<u>DomainCombiner</u> is used to update and optimize <u>ProtectionDomains</u> from an <u>AccessControlContext</u> .
<u>Guard</u>	This interface represents a guard, which is an object that is used to protect access to another object.	<u>Guard</u>	<u>Guard</u> implementors protect access to other objects.
<u>Key</u>	The Key interface is the top-level interface for all keys.	<u>Key</u>	<u>Key</u> is the common interface for all keys.
<u>KeyStore.Entry</u>	A marker interface for KeyStore entry types.	<u>KeyStore.Entry</u>	<u>Entry</u> is the common marker interface for a <u>KeyStore</u> entry.
<u>KeyStore.LoadStoreParameter</u>	A marker interface for KeyStore <u>load</u> and <u>store</u> parameters.	<u>KeyStore.LoadStoreParameter</u>	<u>LoadStoreParameter</u> represents a parameter that specifies how a <u>KeyStore</u> can be loaded and stored.
<u>KeyStore.ProtectionParameter</u>	A marker interface for keystore protection parameters.	<u>KeyStore.ProtectionParameter</u>	<u>ProtectionParameter</u> is a marker interface for protection parameters.
<u>Principal</u>	This interface represents the abstract notion of a principal, which can be used to	<u>Policy.Parameters</u>	A marker interface for Policy parameters.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
	represent any entity, such as an individual, a corporation, and a login id.	Principal	Principals are objects which have identities.
PrivateKey	A private key.	PrivateKey	PrivateKey is the common interface for private keys.
PrivilegedAction<T>	A computation to be performed with privileges enabled.	PrivilegedAction<T>	PrivilegedAction represents an action that can be executed privileged regarding access control.
PrivilegedExceptionAction<T>	A computation to be performed with privileges enabled, that throws one or more checked exceptions.	PrivilegedExceptionAction<T>	PrivilegedAction represents an action, that can be executed privileged regarding access control.
PublicKey	A public key.	PublicKey	PublicKey is the common interface for public keys.
Class Summary		Classes	
AccessControlContext	An AccessControlContext is used to make system resource access decisions based on the context it encapsulates.	AccessControlContext	AccessControlContext encapsulates the ProtectionDomains on which access control decisions are based.
		AccessController	AccessController provides static methods to perform access control checks and privileged operations.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>AlgorithmParameterGenerator</u>	The AlgorithmParameterGenerator class is used to generate a set of parameters to be used with a certain algorithm.	AlgorithmParameterGenerator	AlgorithmParameterGenerator is an engine class which is capable of generating parameters for the algorithm it was initialized with.
<u>AlgorithmParameterGeneratorSpi</u>	This class defines the <i>Service Provider Interface (SPI)</i> for the AlgorithmParameterGenerator class, which is used to generate a set of parameters to be used with a certain algorithm.	AlgorithmParameterGeneratorSpi	AlgorithmParameterGeneratorSpi is the Service Provider Interface (SPI) definition for AlgorithmParameterGenerator.
<u>AlgorithmParameters</u>	This class is used as an opaque representation of cryptographic parameters.	AlgorithmParameters	AlgorithmParameters is an engine class which provides algorithm parameters.
<u>AlgorithmParametersSpi</u>	This class defines the <i>Service Provider Interface (SPI)</i> for the AlgorithmParameters class, which is used to manage algorithm parameters.	AlgorithmParametersSpi	AlgorithmParametersSpi is the Service Provider Interface (SPI) definition for AlgorithmParameters.
<u>AllPermission</u>	The AllPermission is a permission that implies all other permissions.	AllPermission	AllPermission represents the permission to perform any operation.
<u>AuthProvider</u>	This class defines login and logout methods for a provider.	AuthProvider	AuthProvider is an abstract superclass for Java Security Provider which provide login and logout.
<u>BasicPermission</u>	The BasicPermission class extends the Permission class, and can be used as the base class for permissions that want to follow the same naming convention as BasicPermission.	BasicPermission	BasicPermission is the common base class of all permissions which have a name but no action lists.
<u>CodeSigner</u>	This class encapsulates information about a code signer.	CodeSigner	CodeSigner represents a signer of code.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>CodeSource</u>	This class extends the concept of a codebase to encapsulate not only the location (URL) but also the certificate chains that were used to verify signed code originating from that location.	CodeSource	CodeSource encapsulates the location from where code is loaded and the certificates that were used to verify that code.
<u>DigestInputStream</u>	A transparent stream that updates the associated message digest using the bits going through the stream.	DigestInputStream	DigestInputStream is a FilterInputStream which maintains an associated message digest.
<u>DigestOutputStream</u>	A transparent stream that updates the associated message digest using the bits going through the stream.	DigestOutputStream	DigestOutputStream is a FilterOutputStream which maintains an associated message digest.
<u>GuardedObject</u>	A GuardedObject is an object that is used to protect access to another object.	GuardedObject	GuardedObject controls access to an object, by checking all requests for the object with a Guard .
<u>Identity</u>	Deprecated. <i>This class is no longer used.</i>	Identity	<i>This class is deprecated. The functionality of this class has been replace by Principal, KeyStore and the java.security.cert package.</i>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>IdentityScope</u>	Deprecated. <i>This class is no longer used.</i>	<u>IdentityScope</u>	<i>This class is deprecated. The functionality of this class has been replaced by <u>Principal</u>, <u>KeyStore</u> and the <u>java.security.cert</u> package.</i>
<u>KeyFactory</u>	Key factories are used to convert <i>keys</i> (opaque cryptographic keys of type <code>Key</code>) into <i>key specifications</i> (transparent representations of the underlying key material), and vice versa.	<u>KeyFactory</u>	<u>KeyFactory</u> is an engine class that can be used to translate between public and private key objects and convert keys between their external representation, that can be easily transported and their internal representation.
<u>KeyFactorySpi</u>	This class defines the <i>Service Provider Interface (SPI)</i> for the <code>KeyFactory</code> class.	<u>KeyFactorySpi</u>	<u>KeyFactorySpi</u> is the Service Provider Interface (SPI) definition for <u>KeyFactory</u> .
<u>KeyPair</u>	This class is a simple holder for a key pair (a public key and a private key).	<u>KeyPair</u>	<u>KeyPair</u> is a container for a public key and a private key.
<u>KeyPairGenerator</u>	The <code>KeyPairGenerator</code> class is used to generate pairs of public and private keys.	<u>KeyPairGenerator</u>	<u>KeyPairGenerator</u> is an engine class which is capable of generating a private key and its related public key utilizing the algorithm it was initialized with.
<u>KeyPairGeneratorSpi</u>	This class defines the <i>Service Provider Interface (SPI)</i> for the <code>KeyPairGenerator</code> class, which is used to generate pairs of public and private keys.	<u>KeyPairGeneratorSpi</u>	<u>KeyPairGeneratorSpi</u> is the Service Provider Interface (SPI) definition for <u>KeyPairGenerator</u> .
<u>KeyRep</u>	Standardized representation for serialized <code>Key</code> objects.	<u>KeyRep</u>	<u>KeyRep</u> is a standardized representation

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>KeyStore</u>	This class represents a storage facility for cryptographic keys and certificates.		for serialized <u>key</u> objects.
<u>KeyStore.Builder</u>	A description of a to-be-instantiated KeyStore object.	<u>KeyStore</u>	<u>KeyStore</u> is responsible for maintaining cryptographic keys and their owners.
<u>KeyStore.CallbackHandlerProtection</u>	A ProtectionParameter encapsulating a CallbackHandler.	<u>KeyStore.Builder</u>	<u>Builder</u> is used to construct new instances of <u>KeyStore</u> .
<u>KeyStore.PasswordProtection</u>	A password-based implementation of ProtectionParameter.	<u>KeyStore.CallbackHandlerProtection</u>	<u>CallbackHandlerProtection</u> is a <u>ProtectionParameter</u> that encapsulates a <u>CallbackHandler</u> .
<u>KeyStore.PrivateKeyEntry</u>	A KeyStore entry that holds a PrivateKey and corresponding certificate chain.	<u>KeyStore.PasswordProtection</u>	<u>PasswordProtection</u> is a <u>ProtectionParameter</u> that protects a <u>KeyStore</u> using a password.
<u>KeyStore.SecretKeyEntry</u>	A KeyStore entry that holds a SecretKey.	<u>KeyStore.PrivateKeyEntry</u>	<u>PrivateKeyEntry</u> represents a <u>KeyStore</u> entry that holds a private key.
<u>KeyStore.TrustedCertificateEntry</u>	A KeyStore entry that holds a trusted Certificate.	<u>KeyStore.SecretKeyEntry</u>	<u>SecretKeyEntry</u> represents a <u>KeyStore</u> entry that holds a secret key.
		<u>KeyStore.TrustedCertificateEntry</u>	<u>TrustedCertificateEntry</u> represents a <u>KeyStore</u> entry that holds a trusted certificate.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>KeyStoreSpi</u>	This class defines the <i>Service Provider Interface (SPI)</i> for the <code>KeyStore</code> class.	<u>KeyStoreSpi</u>	<u>KeyStoreSpi</u> is the Service Provider Interface (SPI) definition for <u>KeyStore</u> .
<u>MessageDigest</u>	This <code>MessageDigest</code> class provides applications the functionality of a message digest algorithm, such as MD5 or SHA.	<u>MessageDigest</u>	Uses a one-way hash function to turn an arbitrary number of bytes into a fixed-length byte sequence.
<u>MessageDigestSpi</u>	This class defines the <i>Service Provider Interface (SPI)</i> for the <code>MessageDigest</code> class, which provides the functionality of a message digest algorithm, such as MD5 or SHA.	<u>MessageDigestSpi</u>	<u>MessageDigestSpi</u> is the Service Provider Interface (SPI) definition for <u>MessageDigest</u> .
<u>Permission</u>	Abstract class for representing access to a system resource.	<u>Permission</u>	<u>Permission</u> is the common base class of all permissions that participate in the access control security framework around <u>AccessController</u> and <u>AccessControlContext</u> .
<u>PermissionCollection</u>	Abstract class representing a collection of <code>Permission</code> objects.	<u>PermissionCollection</u>	<u>PermissionCollection</u> is the common base class for all collections that provide a convenient method for determining whether or not a given permission is implied by any of the permissions present in this collection.
<u>Permissions</u>	This class represents a heterogeneous collection of <code>Permissions</code> .	<u>Permissions</u>	<u>Permissions</u> represents a <u>PermissionCollection</u> where the contained permissions can be of different types.
		<u>Policy</u>	<u>Policy</u> is the common super type of classes which represent a system security policy.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Policy	This is an abstract class for representing the system security policy for a Java application environment (specifying which permissions are available for code from various sources).	PolicySpi	Represents the Service Provider Interface (SPI) for java.security.Policy class.
ProtectionDomain	This ProtectionDomain class encapsulates the characteristics of a domain, which encloses a set of classes whose instances are granted a set of permissions when being executed on behalf of a given set of Principals.	ProtectionDomain	ProtectionDomain represents all permissions that are granted to a specific code source.
Provider	This class represents a "provider" for the Java Security API, where a provider implements some or all parts of Java Security.	Provider	Provider is the abstract superclass for all security providers in the Java security infrastructure.
Provider.Service	The description of a security service.	Provider.Service	Service represents a service in the Java Security infrastructure.
SecureClassLoader	This class extends ClassLoader with additional support for defining classes with an associated code source and permissions which are retrieved by the system policy by default.	SecureClassLoader	SecureClassLoader represents a ClassLoader which associates the classes it loads with a code source and provide mechanisms to allow the relevant permissions to be retrieved.
SecureRandom	This class provides a cryptographically strong random number generator (RNG).	SecureRandom	This class generates cryptographically secure pseudo-random numbers.
SecureRandomSpi	This class defines the <i>Service Provider Interface (SPI)</i> for the <code>SecureRandom</code> class.	SecureRandomSpi	SecureRandomSpi is the <i>Service Provider Interface (SPI)</i> definition for SecureRandom .
		Security	Security is the central class in the Java Security API.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Security	This class centralizes all security properties and common security methods.	SecurityPermission	SecurityPermission objects guard access to the mechanisms which implement security.
SecurityPermission	This class is for security permissions.	Signature	Signature is an engine class which is capable of creating and verifying digital signatures, using different algorithms that have been registered with the Security class.
Signature	This Signature class is used to provide applications the functionality of a digital signature algorithm.	SignatureSpi	SignatureSpi is the <i>Service Provider Interface (SPI)</i> definition for Signature .
SignatureSpi	This class defines the <i>Service Provider Interface (SPI)</i> for the Signature class, which is used to provide the functionality of a digital signature algorithm.	SignedObject	A SignedObject instance acts as a container for another object.
SignedObject	SignedObject is a class for the purpose of creating authentic runtime objects whose integrity cannot be compromised without being detected.	Signer	<i>This class is deprecated. Replaced by behavior in java.security.cert package and Principal</i>
Signer	Deprecated. <i>This class is no longer used.</i>	Timestamp	Timestamp represents a signed time stamp.
Timestamp	This class encapsulates information about a signed timestamp.	UnresolvedPermission	An UnresolvedPermission represents a Permission whose type should be resolved lazy and not during initialization time of the Policy .
UnresolvedPermission	The UnresolvedPermission class is used to hold Permissions that were "unresolved" when the Policy was initialized.		

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Enum Summary		Enums	
KeyRep.Type	Key type.	KeyRep.Type	Type enumerates the supported key types.
Exception Summary		Exceptions	
AccessControlException	This exception is thrown by the AccessController to indicate that a requested access (to a critical system resource such as the file system or the network) is denied.	AccessControlException	AccessControlException is thrown if the access control infrastructure denies protected access due to missing permissions.
DigestException	This is the generic Message Digest exception.	DigestException	DigestException is a general message digest exception.
GeneralSecurityException	The GeneralSecurityException class is a generic security exception class that provides type safety for all the security-related exception classes that extend from it.	GeneralSecurityException	GeneralSecurityException is a general security exception and the superclass for all security specific exceptions.
InvalidAlgorithmParameterException	This is the exception for invalid or inappropriate algorithm parameters.	InvalidAlgorithmParameterException	InvalidAlgorithmParameterException indicates the occurrence of invalid algorithm parameters.
InvalidKeyException	This is the exception for invalid Keys (invalid encoding, wrong length, uninitialized, etc).	InvalidKeyException	InvalidKeyException indicates exceptional conditions, caused by an invalid key.
InvalidParameterException	This exception, designed for use by the JCA/JCE engine classes, is thrown when		

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>KeyException</u>	This is the basic key exception.	<u>InvalidParameterException</u>	<u>InvalidParameterException</u> indicates exceptional conditions, caused by invalid parameters.
<u>KeyManagementException</u>	This is the general key management exception for all operations dealing with key management.	<u>KeyException</u>	<u>KeyException</u> is the common superclass of all key related exceptions.
<u>KeyStoreException</u>	This is the generic KeyStore exception.	<u>KeyManagementException</u>	<u>KeyManagementException</u> is a general exception, thrown to indicate an exception during processing an operation concerning key management.
<u>NoSuchAlgorithmException</u>	This exception is thrown when a particular cryptographic algorithm is requested but is not available in the environment.	<u>KeyStoreException</u>	<u>KeyStoreException</u> is a general <u>KeyStore</u> exception.
<u>NoSuchProviderException</u>	This exception is thrown when a particular security provider is requested but is not available in the environment.	<u>NoSuchAlgorithmException</u>	<u>NoSuchAlgorithmException</u> indicates that a requested algorithm could not be found.
<u>PrivilegedActionException</u>	This exception is thrown by <code>doPrivileged(PrivilegedExceptionAction)</code> and <code>doPrivileged(PrivilegedExceptionAction, AccessControlContext context)</code> to indicate that the action being performed threw a checked exception.	<u>NoSuchProviderException</u>	<u>NoSuchProviderException</u> indicates that a requested security provider could not be found.
<u>ProviderException</u>	A runtime exception for Provider exceptions (such as misconfiguration errors or unrecoverable internal errors), which may be subclassed by Providers to throw specialized, provider-specific runtime	<u>PrivilegedActionException</u>	<u>PrivilegedActionException</u> wraps exceptions which are thrown from within privileged operations.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
<u>SignatureException</u>	This is the generic Signature exception.	<u>ProviderException</u>	<u>ProviderException</u> is a general exception, thrown by security <u>Providers</u> .
<u>UnrecoverableEntryException</u>	This exception is thrown if an entry in the keystore cannot be recovered.	<u>SignatureException</u>	<u>SignatureException</u> is a general <u>Signature</u> exception.
<u>UnrecoverableKeyException</u>	This exception is thrown if a key in the keystore cannot be recovered.	<u>UnrecoverableEntryException</u>	<u>UnrecoverableEntryException</u> indicates, that a <u>KeyStore.Entry</u> cannot be recovered from a <u>KeyStore</u> .
		<u>UnrecoverableKeyException</u>	<u>UnrecoverableKeyException</u> indicates, that a key cannot be recovered from a <u>KeyStore</u> .

Exhibit Copyright-E

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security.KeyPair)	Android APIs (java.security.KeyPair)
<p>java.security</p> <h3>Class KeyPair</h3> <p>java.lang.Object</p> <p>└ java.security.KeyPair</p> <p>All Implemented Interfaces:</p> <p>Serializable</p> <hr/> <p>public final class KeyPair extends Object implements Serializable</p> <p>This class is a simple holder for a key pair (a public key and a private key). It does not enforce any security, and, when initialized, should be treated like a <code>PrivateKey</code>.</p> <p>See Also:</p> <p>PublicKey, PrivateKey, Serialized Form</p>	<p>public final class</p> <h3>KeyPair</h3> <p>extends Object implements Serializable java.lang.Object ↳ java.security.KeyPair</p> <hr/> <h3>Class Overview</h3> <p><code>KeyPair</code> is a container for a public key and a private key. Since the private key can be accessed, instances must be treated like a private key.</p> <p>See Also</p> <ul style="list-style-type: none"> • PrivateKey • PublicKey
<h3>Constructor Summary</h3> <div> <p>KeyPair(PublicKey publicKey, PrivateKey privateKey)</p> <p>Constructs a key pair from the given public key and private key.</p> </div>	<hr/> <h3>Public Constructors</h3> <div> <p>public KeyPair (PublicKey publicKey, PrivateKey privateKey)</p> <p>Since: API Level 1</p> <p>Constructs a new instance of <code>KeyPair</code> with a public key and the corresponding private key.</p> <p>Parameters</p> </div>

	<div><div>publicKey</div><div>the public key.</div></div> <div><div>privateKey</div><div>the private key.</div></div>								
<div><div>Method Summary</div><table><tr><td>PrivateKey</td><td><div><div>getPrivate()</div><div>Returns a reference to the private key component of this key pair.</div></div></td></tr><tr><td>PublicKey</td><td><div><div>getPublic()</div><div>Returns a reference to the public key component of this key pair.</div></div></td></tr></table><div><div>Methods inherited from class java.lang.Object</div><div>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</div></div></div>	PrivateKey	<div><div>getPrivate()</div><div>Returns a reference to the private key component of this key pair.</div></div>	PublicKey	<div><div>getPublic()</div><div>Returns a reference to the public key component of this key pair.</div></div>	<div><div>Summary</div><div><div>Public Constructors</div><div><div><div>KeyPair(PublicKey publicKey, PrivateKey privateKey)</div><div>Constructs a new instance of KeyPair with a public key and the corresponding private key.</div></div></div><div><div>Public Methods</div><table><tr><td>PrivateKey</td><td><div><div>getPrivate()</div><div>Returns the private key.</div></div></td></tr><tr><td>PublicKey</td><td><div><div>getPublic()</div><div>Returns the public key.</div></div></td></tr></table><div><div>[Expand]</div><div><div>Inherited Methods</div><div><div>►From class java.lang.Object</div></div></div></div></div></div></div>	PrivateKey	<div><div>getPrivate()</div><div>Returns the private key.</div></div>	PublicKey	<div><div>getPublic()</div><div>Returns the public key.</div></div>
PrivateKey	<div><div>getPrivate()</div><div>Returns a reference to the private key component of this key pair.</div></div>								
PublicKey	<div><div>getPublic()</div><div>Returns a reference to the public key component of this key pair.</div></div>								
PrivateKey	<div><div>getPrivate()</div><div>Returns the private key.</div></div>								
PublicKey	<div><div>getPublic()</div><div>Returns the public key.</div></div>								

Constructor Detail

KeyPair

```
public KeyPair(PublicKey publicKey,
               PrivateKey privateKey)
```

Constructs a key pair from the given public key and private key.

Note that this constructor only stores references to the public and private key components in the generated key pair. This is safe, because `Key` objects are immutable.

Parameters:

`publicKey` - the public key.

`privateKey` - the private key.

Public Constructors

```
public KeyPair(PublicKey publicKey, PrivateKey privateKey)
```

Since: API Level 1

Constructs a new instance of `KeyPair` with a public key and the corresponding private key.

Parameters

publicKey the public key.

privateKey the private key.

Method Detail

getPublic

```
public PublicKey getPublic()
```

Returns a reference to the public key component of this key pair.

Returns:

a reference to the public key.

getPrivate

```
public PrivateKey getPrivate()
```

Returns a reference to the private key component of this key pair.

Returns:

Public Methods

```
public PrivateKey getPrivate()
```

Since: API Level 1

Returns the private key.

Returns

• the private key.

```
public PublicKey getPublic()
```

Since: API Level 1

Returns the public key.

a reference to the private key.	Returns
	<ul style="list-style-type: none">the public key.

Exhibit Copyright-F

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)								
<p>java lang</p> <h3>Class Runtime</h3> <p>java.lang.Object</p> <p>└─ java.lang.Runtime</p> <hr/> <pre>public class Runtime extends Object</pre> <p>Every Java application has a single instance of class <code>Runtime</code> that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the <code>getRuntime</code> method.</p> <p>An application cannot create its own instance of this class.</p> <p>Since: JDK1.0</p> <p>See Also: getRuntime()</p>	<p>public class</p> <h3>Runtime</h3> <p>extends Object</p> <p>java.lang.Object</p> <p>↳ java.lang.Runtime</p> <hr/> <h3>Class Overview</h3> <p>Allows Java applications to interface with the environment in which they are running. Applications can not create an instance of this class, but they can get a singleton instance by invoking getRuntime().</p> <p>See Also</p> <p>System</p>								
<h3>Method Summary</h3> <table border="1"> <tr> <td>void</td><td>addShutdownHook(Thread hook) Registers a new virtual-machine shutdown hook.</td></tr> <tr> <td>int</td><td>availableProcessors() Returns the number of processors available to the Java virtual machine.</td></tr> </table>	void	addShutdownHook (Thread hook) Registers a new virtual-machine shutdown hook.	int	availableProcessors () Returns the number of processors available to the Java virtual machine.	<h3>Summary</h3> <p>Public Methods</p> <table border="1"> <tr> <td>void</td><td>addShutdownHook(Thread hook) Registers a virtual-machine shutdown hook.</td></tr> <tr> <td>int</td><td>availableProcessors() Returns the number of processors available to the virtual machine.</td></tr> </table>	void	addShutdownHook (Thread hook) Registers a virtual-machine shutdown hook.	int	availableProcessors () Returns the number of processors available to the virtual machine.
void	addShutdownHook (Thread hook) Registers a new virtual-machine shutdown hook.								
int	availableProcessors () Returns the number of processors available to the Java virtual machine.								
void	addShutdownHook (Thread hook) Registers a virtual-machine shutdown hook.								
int	availableProcessors () Returns the number of processors available to the virtual machine.								

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)		Android APIs (java.lang.Runtime)	
Process	exec (String command) Executes the specified string command in a separate process.	Process	exec (String [] progArray, String [] envp) Executes the specified command and its arguments in a separate native process.
Process	exec (String [] cmdarray) Executes the specified command and arguments in a separate process.	Process	exec (String prog, String [] envp, File directory) Executes the specified program in a separate native process.
Process	exec (String [] cmdarray, String [] envp) Executes the specified command and arguments in a separate process with the specified environment.	Process	exec (String [] progArray, String [] envp, File directory) Executes the specified command and its arguments in a separate native process.
Process	exec (String [] cmdarray, String [] envp, File dir) Executes the specified command and arguments in a separate process with the specified environment and working directory.	Process	exec (String prog, String [] envp) Executes the specified program in a separate native process.
Process	exec (String command, String [] envp) Executes the specified string command in a separate process with the specified environment.	Process	exec (String prog) Executes the specified program in a separate native process.
Process	exec (String command, String [] envp, File dir) Executes the specified string command in a separate process with the specified environment and working directory.	Process	exec (String [] progArray) Executes the specified command and its arguments in a separate native process.
void	exit (int status) Terminates the currently running Java virtual machine by initiating its shutdown sequence.	void	exit (int code) Causes the virtual machine to stop running and the program to exit.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)		Android APIs (java.lang.Runtime)	
long	freeMemory() Returns the amount of free memory in the Java Virtual Machine.	long	freeMemory() Returns the amount of free memory resources which are available to the running program.
void	gc() Runs the garbage collector.	void	gc() Indicates to the virtual machine that it would be a good time to run the garbage collector.
InputStream	getLocalizedInputStream(InputStream in) Deprecated. As of JDK 1.1, the preferred way to translate a byte stream in the local encoding into a character stream in Unicode is via the <i>InputStreamReader</i> and <i>BufferedReader</i> classes.	InputStream	getLocalizedInputStream(InputStream stream) <i>This method is deprecated. Use InputStreamReader.</i>
OutputStream	getLocalizedOutputStream(OutputStream out) Deprecated. As of JDK 1.1, the preferred way to translate a Unicode character stream into a byte stream in the local encoding is via the <i>OutputStreamWriter</i> , <i>BufferedWriter</i> , and <i>PrintWriter</i> classes.	OutputStream	getLocalizedOutputStream(OutputStream stream) <i>This method is deprecated. Use OutputStreamWriter.</i>
static Runtime	getRuntime() Returns the runtime object associated with the current Java application.	static Runtime	getRuntime() Returns the single Runtime instance.
void	halt(int status) Forcibly terminates the currently running Java virtual machine.	void	halt(int code) Causes the virtual machine to stop running, and the program to exit.
void	load(String filename) Loads the specified filename as a dynamic library.	void	load(String pathName) Loads and links the dynamic library that is identified through the specified path.
void	loadLibrary(String libname) Loads the dynamic library with the specified library name.	void	loadLibrary(String libName) Loads and links the library with the specified name.
		long	maxMemory()

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)		Android APIs (java.lang.Runtime)	
long	maxMemory() Returns the maximum amount of memory that the Java virtual machine will attempt to use.		Returns the maximum amount of memory that may be used by the virtual machine, or <code>Long.MAX_VALUE</code> if there is no such limit.
boolean	removeShutdownHook(Thread hook) De-registers a previously-registered virtual-machine shutdown hook.	boolean	removeShutdownHook(Thread hook) Unregisters a previously registered virtual machine shutdown hook.
void	runFinalization() Runs the finalization methods of any objects pending finalization.	void	runFinalization() Provides a hint to the virtual machine that it would be useful to attempt to perform any outstanding object finalization.
static void	runFinalizersOnExit(boolean value) Deprecated. <i>This method is inherently unsafe. It may result in finalizers being called on live objects while other threads are concurrently manipulating those objects, resulting in erratic behavior or deadlock.</i>	static void	runFinalizersOnExit(boolean run) <i>This method is deprecated. This method is unsafe.</i>
long	totalMemory() Returns the total amount of memory in the Java virtual machine.	long	totalMemory() Returns the total amount of memory which is available to the running program.
void	traceInstructions(boolean on) Enables/Disables tracing of instructions.	void	traceInstructions(boolean enable) Switches the output of debug information for instructions on or off.
void	traceMethodCalls(boolean on) Enables/Disables tracing of method calls.	void	traceMethodCalls(boolean enable) Switches the output of debug information for methods on or off.
Methods inherited from class java.lang.Object clone , equals , finalize , getClass , hashCode , notify , notifyAll , toString , wait , wait , wait		Inherited Methods^[1]	

¹ Collapsed view.

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)																
	<p>►From class java.lang.Object</p>																
	<p>Inherited Methods^[2]</p> <p>▼From class java.lang.Object</p> <table border="1"> <tbody> <tr> <td data-bbox="1226 483 1549 521">Objectclone()</td><td data-bbox="1549 483 1969 589">Creates and returns a copy of this Object.</td></tr> <tr> <td data-bbox="1226 589 1549 626">boolean equals(Object o)</td><td data-bbox="1549 589 1969 727">Compares this instance with the specified object and indicates if they are equal.</td></tr> <tr> <td data-bbox="1226 727 1549 764">void finalize()</td><td data-bbox="1549 727 1969 865">Called before the object's memory is reclaimed by the VM.</td></tr> <tr> <td data-bbox="1226 865 1549 902">final getClass()</td><td data-bbox="1549 865 1969 1003">Returns the unique instance of Class that represents this object's class.</td></tr> <tr> <td data-bbox="1226 902 1549 940">Class<? extends Object> hashCode()</td><td data-bbox="1549 902 1969 1040">Returns an integer hash code for this object.</td></tr> <tr> <td data-bbox="1226 1040 1549 1078">int hashCode()</td><td data-bbox="1549 1040 1969 1102">Returns an integer hash code for this object.</td></tr> <tr> <td data-bbox="1226 1102 1549 1140">final void notify()</td><td data-bbox="1549 1102 1969 1312">Causes a thread which is waiting on this object's monitor (by means of calling one of the wait() methods) to be woken up.</td></tr> <tr> <td data-bbox="1226 1312 1549 1349">final void notifyAll()</td><td data-bbox="1549 1312 1969 1450">Causes all threads which are waiting on this object's monitor (by means of calling one of the</td></tr> </tbody> </table>	Object clone()	Creates and returns a copy of this Object .	boolean equals(Object o)	Compares this instance with the specified object and indicates if they are equal.	void finalize()	Called before the object's memory is reclaimed by the VM.	final getClass()	Returns the unique instance of Class that represents this object's class.	Class <? extends Object > hashCode()	Returns an integer hash code for this object.	int hashCode()	Returns an integer hash code for this object.	final void notify()	Causes a thread which is waiting on this object's monitor (by means of calling one of the wait() methods) to be woken up.	final void notifyAll()	Causes all threads which are waiting on this object's monitor (by means of calling one of the
Object clone()	Creates and returns a copy of this Object .																
boolean equals(Object o)	Compares this instance with the specified object and indicates if they are equal.																
void finalize()	Called before the object's memory is reclaimed by the VM.																
final getClass()	Returns the unique instance of Class that represents this object's class.																
Class <? extends Object > hashCode()	Returns an integer hash code for this object.																
int hashCode()	Returns an integer hash code for this object.																
final void notify()	Causes a thread which is waiting on this object's monitor (by means of calling one of the wait() methods) to be woken up.																
final void notifyAll()	Causes all threads which are waiting on this object's monitor (by means of calling one of the																

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)										
	<table border="1"> <tr> <td></td><td><code>wait()</code> methods) to be woken up.</td></tr> <tr> <td><code>String toString()</code></td><td>Returns a string containing a concise, human-readable description of this object.</td></tr> <tr> <td>final void <code>wait()</code></td><td>Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object.</td></tr> <tr> <td>final void <code>wait(long millis, int nanos)</code></td><td>Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object or until the specified timeout expires.</td></tr> <tr> <td>final void <code>wait(long millis)</code></td><td>Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object or until the specified timeout expires.</td></tr> </table>		<code>wait()</code> methods) to be woken up.	<code>String toString()</code>	Returns a string containing a concise, human-readable description of this object.	final void <code>wait()</code>	Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object.	final void <code>wait(long millis, int nanos)</code>	Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object or until the specified timeout expires.	final void <code>wait(long millis)</code>	Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object or until the specified timeout expires.
	<code>wait()</code> methods) to be woken up.										
<code>String toString()</code>	Returns a string containing a concise, human-readable description of this object.										
final void <code>wait()</code>	Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object.										
final void <code>wait(long millis, int nanos)</code>	Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object or until the specified timeout expires.										
final void <code>wait(long millis)</code>	Causes the calling thread to wait until another thread calls the <code>notify()</code> or <code>notifyAll()</code> method of this object or until the specified timeout expires.										

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
Method Detail	Public Methods
<p>addShutdownHook</p> <pre>public void addShutdownHook(Thread hook)</pre> <p>Registers a new virtual-machine shutdown hook.</p> <p>The Java virtual machine <i>shuts down</i> in response to two kinds of events:</p> <ul style="list-style-type: none"> • The program <i>exits</i> normally, when the last non-daemon thread exits or when the exit (equivalently, System.exit) method is invoked, or • The virtual machine is <i>terminated</i> in response to a user interrupt, such as typing ^C, or a system-wide event, such as user logoff or system shutdown. <p>A <i>shutdown hook</i> is simply an initialized but unstarted thread. When the virtual machine begins its shutdown sequence it will start all registered shutdown hooks in some unspecified order and let them run concurrently. When all the hooks have finished it will then run all uninvoked finalizers if finalization-on-exit has been enabled. Finally, the virtual machine will halt. Note that daemon threads will continue to run during the shutdown sequence, as will non-daemon threads if shutdown was initiated by invoking the exit method.</p> <p>Once the shutdown sequence has begun it can be stopped only by invoking the halt method, which forcibly terminates the virtual machine.</p> <p>Once the shutdown sequence has begun it is impossible to register a new shutdown hook or de-register a previously-registered hook. Attempting either of these operations will cause an IllegalStateException to be thrown.</p>	<pre>public void addShutdownHook(Thread hook)</pre> <p>Since: API Level 1</p> <p>Registers a virtual-machine shutdown hook. A shutdown hook is a Thread that is ready to run, but has not yet been started. All registered shutdown hooks will be executed once the virtual machine shuts down properly. A proper shutdown happens when either the exit(int) method is called or the surrounding system decides to terminate the application, for example in response to a CTRL-C or a system-wide shutdown. A termination of the virtual machine due to the halt(int) method, an Error or a SIGKILL, in contrast, is not considered a proper shutdown. In these cases the shutdown hooks will not be run.</p> <p>Shutdown hooks are run concurrently and in an unspecified order. Hooks failing due to an unhandled exception are not a problem, but the stack trace might be printed to the console. Once initiated, the whole shutdown process can only be terminated by calling <code>halt()</code>.</p> <p>If runFinalizersOnExit(boolean) has been called with a <code>true</code> argument, garbage collection and finalization will take place after all hooks are either finished or have failed. Then the virtual machine terminates.</p> <p>It is recommended that shutdown hooks do not do any time-consuming activities, in order to not hold up the shutdown process longer than necessary.</p> <p>Parameters</p> <p><i>hook</i> the shutdown hook to register.</p> <p>Throws</p> <p>IllegalArgumentExeption if the hook has already been started or if it has already been registered.</p> <p>IllegalStateException if the virtual machine is already shutting</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>Shutdown hooks run at a delicate time in the life cycle of a virtual machine and should therefore be coded defensively. They should, in particular, be written to be thread-safe and to avoid deadlocks insofar as possible. They should also not rely blindly upon services that may have registered their own shutdown hooks and therefore may themselves in the process of shutting down.</p> <p>Shutdown hooks should also finish their work quickly. When a program invokes exit the expectation is that the virtual machine will promptly shut down and exit. When the virtual machine is terminated due to user logoff or system shutdown the underlying operating system may only allow a fixed amount of time in which to shut down and exit. It is therefore inadvisable to attempt any user interaction or to perform a long-running computation in a shutdown hook.</p> <p>Uncaught exceptions are handled in shutdown hooks just as in any other thread, by invoking the uncaughtException method of the thread's ThreadGroup object. The default implementation of this method prints the exception's stack trace to System.err and terminates the thread; it does not cause the virtual machine to exit or halt.</p> <p>In rare circumstances the virtual machine may <i>abort</i>, that is, stop running without shutting down cleanly. This occurs when the virtual machine is terminated externally, for example with the SIGKILL signal on Unix or the TerminateProcess call on Microsoft Windows. The virtual machine may also abort if a native method goes awry by, for example, corrupting internal data structures or attempting to access nonexistent memory. If the virtual machine aborts then no guarantee can be made about whether or not any shutdown hooks will be run.</p> <p>Parameters: hook - An initialized but unstarted Thread object</p> <p>Throws:</p>	<p>down.</p> <p>SecurityException</p> <p>if a SecurityManager is registered and the calling code doesn't have the RuntimePermission("shutdownHooks").</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>IllegalArgumentException - If the specified hook has already been registered, or if it can be determined that the hook is already running or has already been run</p> <p>IllegalStateException - If the virtual machine is already in the process of shutting down</p> <p>SecurityException - If a security manager is present and it denies RuntimePermission("shutdownHooks")</p> <p>Since: 1.3</p> <p>See Also: removeShutdownHook(java.lang.Thread), halt(int), exit(int)</p>	
<p>availableProcessors</p> <p>public int availableProcessors()</p> <p>Returns the number of processors available to the Java virtual machine.</p> <p>This value may change during a particular invocation of the virtual machine. Applications that are sensitive to the number of available processors should therefore occasionally poll this property and adjust their resource usage appropriately.</p> <p>Returns: the maximum number of processors available to the virtual machine; never smaller than one</p> <p>Since: 1.4</p>	<p>public int availableProcessors ()</p> <p>Since: API Level 1</p> <p>Returns the number of processors available to the virtual machine.</p> <p>Returns the number of available processors, at least 1.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>exit</p> <pre>public void exit(int status)</pre> <p>Terminates the currently running Java virtual machine by initiating its shutdown sequence. This method never returns normally. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.</p> <p>The virtual machine's shutdown sequence consists of two phases. In the first phase all registered shutdown hooks, if any, are started in some unspecified order and allowed to run concurrently until they finish. In the second phase all uninvoked finalizers are run if finalization-on-exit has been enabled. Once this is done the virtual machine halts.</p> <p>If this method is invoked after the virtual machine has begun its shutdown sequence then if shutdown hooks are being run this method will block indefinitely. If shutdown hooks have already been run and on-exit finalization has been enabled then this method halts the virtual machine with the given status code if the status is nonzero; otherwise, it blocks indefinitely.</p> <p>The System.exit method is the conventional and convenient means of invoking this method.</p> <p>Parameters: status - Termination status. By convention, a nonzero status code indicates abnormal termination.</p> <p>Throws: SecurityException - If a security manager is present and its checkExit method does not permit exiting with the specified status</p> <p>See Also: SecurityException, SecurityManager.checkExit(int), addShutdownHook(java.lang.Thread), removeShutdownHook(java.lang.Thread),</p>	<pre>public Process exec (String[] progArray, String[] envp)</pre> <p>Since: API Level 1</p> <p>Executes the specified command and its arguments in a separate native process. The new process uses the environment provided in envp. Calling this method is equivalent to calling <code>exec(progArray, envp, null)</code>.</p> <p>Parameters</p> <p><i>progArray</i> the array containing the program to execute as well as any arguments to the program.</p> <p><i>envp</i> the array containing the environment to start the new process in.</p> <p>Returns</p> <p>the new <code>Process</code> object that represents the native process.</p> <p>Throws</p> <p>IOException if the requested program can not be executed.</p> <p>SecurityException if the current <code>SecurityManager</code> disallows program execution.</p> <p>See Also</p> <p>checkExec(String)</p> <pre>public Process exec (String prog, String[] envp, File directory)</pre> <p>Since: API Level 1</p> <p>Executes the specified program in a separate native process. The new process uses the environment provided in envp and the working directory specified by directory.</p> <p>Parameters</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>runFinalizersOnExit(boolean), halt(int)</p> <p>...</p> <hr/> <p>exec</p> <p>public Process exec(String command) throws IOException</p> <p>Executes the specified string command in a separate process.</p> <p>This is a convenience method. An invocation of the form <code>exec(command)</code> behaves in exactly the same way as the invocation <code>exec(command, null, null)</code>.</p> <p>Parameters: command - a specified system command.</p> <p>Returns: A new Process object for managing the subprocess</p> <p>Throws: SecurityException - If a security manager exists and its checkExec method doesn't allow creation of the subprocess IOException - If an I/O error occurs NullPointerException - If command is null IllegalArgumentException - If command is empty</p> <p>See Also: exec(String[], String[], File), ProcessBuilder</p>	<p><i>prog</i> the name of the program to execute.</p> <p><i>envp</i> the array containing the environment to start the new process in.</p> <p><i>directory</i> the directory in which to execute the program. If null, execute if in the same directory as the parent process.</p> <p>Returns the new <code>Process</code> object that represents the native process.</p> <p>Throws IOException if the requested program can not be executed. SecurityException if the current <code>SecurityManager</code> disallows program execution.</p> <p>See Also checkExec(String)</p> <hr/> <p>public Process exec (String[] progArray, String[] envp, File directory)</p> <p>Since: API Level 1</p> <p>Executes the specified command and its arguments in a separate native process. The new process uses the environment provided in <code>envp</code> and the working directory specified by <code>directory</code>.</p> <p>Parameters</p> <p><i>progArray</i> the array containing the program to execute as well as any arguments to the program.</p> <p><i>envp</i> the array containing the environment to start the new process in.</p>
<p>exec</p> <p>public Process exec(String command, String[] envp) throws IOException</p> <p>Executes the specified string command in a separate process with the specified environment.</p>	

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>This is a convenience method. An invocation of the form <code>exec(command, envp)</code> behaves in exactly the same way as the invocation <code>exec(command, envp, null)</code>.</p> <p>Parameters: <code>command</code> - a specified system command. <code>envp</code> - array of strings, each element of which has environment variable settings in the format <i>name=value</i>, or <code>null</code> if the subprocess should inherit the environment of the current process.</p> <p>Returns: A new Process object for managing the subprocess</p> <p>Throws: SecurityException - If a security manager exists and its checkExec method doesn't allow creation of the subprocess IOException - If an I/O error occurs NullPointerException - If <code>command</code> is <code>null</code>, or one of the elements of <code>envp</code> is <code>null</code> IllegalArgumentException - If <code>command</code> is empty</p> <p>See Also: exec(String[], String[], File), ProcessBuilder</p>	<p><i>directory</i> the directory in which to execute the program. If <code>null</code>, execute if in the same directory as the parent process.</p> <p>Returns the new <code>Process</code> object that represents the native process.</p> <p>Throws IOException if the requested program can not be executed. SecurityException if the current <code>SecurityManager</code> disallows program execution.</p> <p>See Also checkExec(String)</p>
<p>exec</p> <pre>public Process exec(String command, String[] envp, File dir) throws IOException</pre> <p>Executes the specified string command in a separate process with the specified environment and working directory.</p> <p>This is a convenience method. An invocation of the form <code>exec(command, envp, dir)</code> behaves in exactly the same way as the invocation <code>exec(cmdarray, envp, dir)</code>, where <code>cmdarray</code> is an array</p>	<pre>public Process exec (String prog, String[] envp)</pre> <p>Since: API Level 1</p> <p>Executes the specified program in a separate native process. The new process uses the environment provided in <code>envp</code>. Calling this method is equivalent to calling <code>exec(prog, envp, null)</code>.</p> <p>Parameters <i>prog</i> the name of the program to execute. <i>envp</i> the array containing the environment to start the new process in.</p> <p>Returns the new <code>Process</code> object that represents the native process.</p> <p>Throws IOException if the requested program can not be executed.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>of all the tokens in <code>command</code>.</p> <p>More precisely, the <code>command</code> string is broken into tokens using a StringTokenizer created by the call <code>new StringTokenizer(command)</code> with no further modification of the character categories. The tokens produced by the tokenizer are then placed in the new string array <code>cmdarray</code>, in the same order.</p> <p>Parameters:</p> <p><code>command</code> - a specified system command.</p> <p><code>envp</code> - array of strings, each element of which has environment variable settings in the format <i>name=value</i>, or <code>null</code> if the subprocess should inherit the environment of the current process.</p> <p><code>dir</code> - the working directory of the subprocess, or <code>null</code> if the subprocess should inherit the working directory of the current process.</p> <p>Returns:</p> <p>A new Process object for managing the subprocess</p> <p>Throws:</p> <p>SecurityException - If a security manager exists and its checkExec method doesn't allow creation of the subprocess</p> <p>IOException - If an I/O error occurs</p> <p>NullPointerException - If <code>command</code> is <code>null</code>, or one of the elements of <code>envp</code> is <code>null</code></p> <p>IllegalArgumentException - If <code>command</code> is empty</p> <p>Since:</p> <p>1.3</p> <p>See Also:</p> <p>ProcessBuilder</p>	<p>SecurityException if the current <code>SecurityManager</code> disallows program execution.</p> <p>See Also</p> <p>checkExec(String)</p> <p>public Process exec (String prog)</p> <p>Since: API Level 1</p> <p>Executes the specified program in a separate native process. The new process inherits the environment of the caller. Calling this method is equivalent to calling <code>exec(prog, null, null)</code>.</p> <p>Parameters</p> <p><code>prog</code> the name of the program to execute.</p> <p>Returns</p> <p>the new <code>Process</code> object that represents the native process.</p> <p>Throws</p> <p>IOException if the requested program can not be executed.</p> <p>SecurityException if the current <code>SecurityManager</code> disallows program execution.</p> <p>See Also</p> <p>checkExec(String)</p> <p>public Process exec (String[] progArray)</p> <p>Since: API Level 1</p> <p>Executes the specified command and its arguments in a separate native process. The new process inherits the environment of the caller. Calling this method is equivalent to calling <code>exec(progArray, null, null)</code>.</p>
<p>exec</p> <p>public Process exec(String[] cmdarray) throws IOException</p> <p>Executes the specified command and arguments in a separate process.</p>	

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>This is a convenience method. An invocation of the form <code>exec(cmdarray)</code> behaves in exactly the same way as the invocation <code>exec(cmdarray, null, null)</code>.</p> <p>Parameters: <code>cmdarray</code> - array containing the command to call and its arguments.</p> <p>Returns: A new Process object for managing the subprocess</p> <p>Throws: SecurityException - If a security manager exists and its checkExec method doesn't allow creation of the subprocess IOException - If an I/O error occurs NullPointerException - If <code>cmdarray</code> is null, or one of the elements of <code>cmdarray</code> is null IndexOutOfBoundsException - If <code>cmdarray</code> is an empty array (has length 0)</p> <p>See Also: ProcessBuilder</p>	<p>Parameters</p> <p><i>progArray</i> the array containing the program to execute as well as any arguments to the program.</p> <p>Returns the new <code>Process</code> object that represents the native process.</p> <p>Throws</p> <p>IOException if the requested program can not be executed.</p> <p>SecurityException if the current <code>SecurityManager</code> disallows program execution.</p> <p>See Also checkExec(String)</p>
<p>exec</p> <pre>public Process exec(String[] cmdarray, String[] envp) throws IOException</pre> <p>Executes the specified command and arguments in a separate process with the specified environment.</p> <p>This is a convenience method. An invocation of the form <code>exec(cmdarray, envp)</code> behaves in exactly the same way as the invocation <code>exec(cmdarray, envp, null)</code>.</p> <p>Parameters: <code>cmdarray</code> - array containing the command to call and its arguments.</p>	<pre>public void exit (int code)</pre> <p>Since: API Level 1</p> <p>Causes the virtual machine to stop running and the program to exit. If runFinalizersOnExit(boolean) has been previously invoked with a <code>true</code> argument, then all objects will be properly garbage-collected and finalized first.</p> <p>Parameters</p> <p><i>code</i> the return code. By convention, non-zero return codes indicate abnormal terminations.</p> <p>Throws</p> <p>SecurityException if the current <code>SecurityManager</code> does not allow the running thread to terminate the virtual machine.</p> <p>See Also</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>envp - array of strings, each element of which has environment variable settings in the format <i>name=value</i>, or null if the subprocess should inherit the environment of the current process.</p> <p>Returns: A new Process object for managing the subprocess</p> <p>Throws: SecurityException - If a security manager exists and its checkExec method doesn't allow creation of the subprocess IOException - If an I/O error occurs NullPointerException - If cmdarray is null, or one of the elements of cmdarray is null, or one of the elements of envp is null IndexOutOfBoundsException - If cmdarray is an empty array (has length 0)</p> <p>See Also: ProcessBuilder</p> <hr/> <p>exec</p> <pre>public Process exec(String[] cmdarray, String[] envp, File dir) throws IOException</pre> <p>Executes the specified command and arguments in a separate process with the specified environment and working directory.</p> <p>Given an array of strings cmdarray, representing the tokens of a command line, and an array of strings envp, representing "environment" variable settings, this method creates a new process in which to execute the specified command.</p> <p>This method checks that cmdarray is a valid operating system command. Which commands are valid is system-dependent, but at the very least the command must be a non-empty list of non-null strings.</p> <p>If envp is null, the subprocess inherits the environment settings of the</p>	<p>checkExit(int)</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>current process.</p> <p>ProcessBuilder.start() is now the preferred way to start a process with a modified environment.</p> <p>The working directory of the new subprocess is specified by <code>dir</code>. If <code>dir</code> is <code>null</code>, the subprocess inherits the current working directory of the current process.</p> <p>If a security manager exists, its checkExec method is invoked with the first component of the array <code>cmdarray</code> as its argument. This may result in a SecurityException being thrown.</p> <p>Starting an operating system process is highly system-dependent. Among the many things that can go wrong are:</p> <ul style="list-style-type: none"> • The operating system program file was not found. • Access to the program file was denied. • The working directory does not exist. <p>In such cases an exception will be thrown. The exact nature of the exception is system-dependent, but it will always be a subclass of IOException.</p> <p>Parameters:</p> <p><code>cmdarray</code> - array containing the command to call and its arguments.</p> <p><code>envp</code> - array of strings, each element of which has environment variable settings in the format <i>name=value</i>, or <code>null</code> if the subprocess should inherit the environment of the current process.</p> <p><code>dir</code> - the working directory of the subprocess, or <code>null</code> if the subprocess should inherit the working directory of the current process.</p> <p>Returns:</p> <p>A new Process object for managing the subprocess</p> <p>Throws:</p>	

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>SecurityException - If a security manager exists and its checkExec method doesn't allow creation of the subprocess</p> <p>IOException - If an I/O error occurs</p> <p>NullPointerException - If cmdarray is null, or one of the elements of cmdarray is null, or one of the elements of envp is null</p> <p>IndexOutOfBoundsException - If cmdarray is an empty array (has length 0)</p> <p>Since: 1.3</p> <p>See Also: ProcessBuilder</p>	

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>freeMemory</p> <p>public long freeMemory()</p> <p>Returns the amount of free memory in the Java Virtual Machine. Calling the <code>gc</code> method may result in increasing the value returned by <code>freeMemory</code>.</p> <p>Returns: an approximation to the total amount of memory currently available for future allocated objects, measured in bytes.</p>	<p>public long freeMemory ()</p> <p>Since: API Level 1</p> <p>Returns the amount of free memory resources which are available to the running program.</p> <p>Returns the approximate amount of free memory, measured in bytes.</p>
<p>gc</p> <p>public void gc()</p> <p>Runs the garbage collector. Calling this method suggests that the Java virtual machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the virtual machine has made its best effort to recycle all discarded objects.</p> <p>The name <code>gc</code> stands for "garbage collector". The virtual machine performs this recycling process automatically as needed, in a separate thread, even if the <code>gc</code> method is not invoked explicitly.</p> <p>The method System.gc() is the conventional and convenient means of invoking this method.</p>	<p>public void gc ()</p> <p>Since: API Level 1</p> <p>Indicates to the virtual machine that it would be a good time to run the garbage collector. Note that this is a hint only. There is no guarantee that the garbage collector will actually be run.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>getLocalizedInputStream</p> <p>@Deprecated public InputStream getLocalizedInputStream(InputStream in)</p> <p>Deprecated. <i>As of JDK 1.1, the preferred way to translate a byte stream in the local encoding into a character stream in Unicode is via the InputStreamReader and BufferedReader classes.</i></p> <p>Creates a localized version of an input stream. This method takes an InputStream and returns an InputStream equivalent to the argument in all respects except that it is localized: as characters in the local character set are read from the stream, they are automatically converted from the local character set to Unicode.</p> <p>If the argument is already a localized stream, it may be returned as the result.</p> <p>Parameters: in - InputStream to localize</p> <p>Returns: a localized input stream</p> <p>See Also: InputStream, BufferedReader, BufferedReader(java.io.Reader), InputStreamReader, InputStreamReader(java.io.InputStream)</p>	<pre>public InputStream getLocalizedInputStream (InputStream stream)</pre> <p>Since: API Level 1</p> <p>This method is deprecated. Use InputStreamReader.</p> <p>Returns the localized version of the specified input stream. The input stream that is returned automatically converts all characters from the local character set to Unicode after reading them from the underlying stream.</p> <p>Parameters</p> <p><i>stream</i> the input stream to localize.</p> <p>Returns</p> <p>the localized input stream.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>getLocalizedOutputStream</p> <p>@Deprecated public OutputStream getLocalizedOutputStream(OutputStream out)</p> <p>Deprecated. <i>As of JDK 1.1, the preferred way to translate a Unicode character stream into a byte stream in the local encoding is via the OutputStreamWriter, BufferedWriter, and PrintWriter classes.</i> Creates a localized version of an output stream. This method takes an OutputStream and returns an OutputStream equivalent to the argument in all respects except that it is localized: as Unicode characters are written to the stream, they are automatically converted to the local character set.</p> <p>If the argument is already a localized stream, it may be returned as the result.</p> <p>Parameters: out - OutputStream to localize</p> <p>Returns: a localized output stream</p> <p>See Also: OutputStream, BufferedWriter.BufferedWriter(java.io.Writer), OutputStreamWriter.OutputStreamWriter(java.io.OutputStream), PrintWriter.PrintWriter(java.io.OutputStream)</p>	<pre>public OutputStream getLocalizedOutputStream (OutputStream stream)</pre> <p>Since: API Level 1</p> <p>This method is deprecated. Use OutputStreamWriter.</p> <p>Returns the localized version of the specified output stream. The output stream that is returned automatically converts all characters from Unicode to the local character set before writing them to the underlying stream.</p> <p>Parameters</p> <p><i>stream</i> the output stream to localize.</p> <p>Returns</p> <p>the localized output stream.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>getRuntime</p> <pre>public static Runtime getRuntime()</pre> <p>Returns the runtime object associated with the current Java application. Most of the methods of class <code>Runtime</code> are instance methods and must be invoked with respect to the current runtime object.</p> <p>Returns: the <code>Runtime</code> object associated with the current Java application.</p>	<pre>public static Runtime getRuntime ()</pre> <p>Since: API Level 1</p> <p>Returns the single <code>Runtime</code> instance.</p> <p>Returns the <code>Runtime</code> object for the current application.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>halt</p> <p>public void halt(int status)</p> <p>Forcibly terminates the currently running Java virtual machine. This method never returns normally.</p> <p>This method should be used with extreme caution. Unlike the exit method, this method does not cause shutdown hooks to be started and does not run uninvoked finalizers if finalization-on-exit has been enabled. If the shutdown sequence has already been initiated then this method does not wait for any running shutdown hooks or finalizers to finish their work.</p> <p>Parameters:</p> <p>status - Termination status. By convention, a nonzero status code indicates abnormal termination. If the exit (equivalently, System.exit) method has already been invoked then this status code will override the status code passed to that method.</p> <p>Throws:</p> <p>SecurityException - If a security manager is present and its checkExit method does not permit an exit with the specified status</p> <p>Since:</p> <p>1.3</p> <p>See Also:</p> <p>exit(int), addShutdownHook(java.lang.Thread), removeShutdownHook(java.lang.Thread)</p>	<p>public void halt (int code)</p> <p>Since: API Level 1</p> <p>Causes the virtual machine to stop running, and the program to exit. Neither shutdown hooks nor finalizers are run before.</p> <p>Parameters</p> <p>code the return code. By convention, non-zero return codes indicate abnormal terminations.</p> <p>Throws</p> <p>SecurityException if the current <code>SecurityManager</code> does not allow the running thread to terminate the virtual machine.</p> <p>See Also</p> <p>checkExit(int)</p> <p>addShutdownHook(Thread)</p> <p>removeShutdownHook(Thread)</p> <p>runFinalizersOnExit(boolean)</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>load</p> <pre>public void load(String filename)</pre> <p>Loads the specified filename as a dynamic library. The filename argument must be a complete path name. From java_g it will automatically insert "_g" before the ".so" (for example <code>Runtime.getRuntime().load("/home/avh/lib/libX11.so");</code>).</p> <p>First, if there is a security manager, its <code>checkLink</code> method is called with the <code>filename</code> as its argument. This may result in a security exception.</p> <p>This is similar to the method loadLibrary(String), but it accepts a general file name as an argument rather than just a library name, allowing any file of native code to be loaded.</p> <p>The method System.load(String) is the conventional and convenient means of invoking this method.</p> <p>Parameters: <code>filename</code> - the file to load.</p> <p>Throws: SecurityException - if a security manager exists and its <code>checkLink</code> method doesn't allow loading of the specified dynamic library UnsatisfiedLinkError - if the file does not exist. NullPointerException - if <code>filename</code> is null</p> <p>See Also: getRuntime(), SecurityException, SecurityManager.checkLink(java.lang.String)</p>	<pre>public void load (String pathName)</pre> <p>Since: API Level 1</p> <p>Loads and links the dynamic library that is identified through the specified path. This method is similar to loadLibrary(String), but it accepts a full path specification whereas <code>loadLibrary</code> just accepts the name of the library to load.</p> <p>Parameters</p> <p><i>pathName</i> the absolute (platform dependent) path to the library to load.</p> <p>Throws</p> <p>UnsatisfiedLinkError if the library can not be loaded.</p> <p>SecurityException if the current <code>SecurityManager</code> does not allow to load the library.</p> <p>See Also</p> <p>checkLink(String)</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>loadLibrary</p> <pre>public void loadLibrary(String libname)</pre> <p>Loads the dynamic library with the specified library name. A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent. The mapping from a library name to a specific filename is done in a system-specific manner.</p> <p>First, if there is a security manager, its <code>checkLink</code> method is called with the <code>libname</code> as its argument. This may result in a security exception.</p> <p>The method System.loadLibrary(String) is the conventional and convenient means of invoking this method. If native methods are to be used in the implementation of a class, a standard strategy is to put the native code in a library file (call it <code>LibFile</code>) and then to put a static initializer:</p> <pre>static { System.loadLibrary("LibFile"); }</pre> <p>within the class declaration. When the class is loaded and initialized, the necessary native code implementation for the native methods will then be loaded as well.</p> <p>If this method is called more than once with the same library name, the second and subsequent calls are ignored.</p> <p>Parameters: <code>libname</code> - the name of the library.</p> <p>Throws: SecurityException - if a security manager exists and its <code>checkLink</code> method doesn't allow loading of the specified dynamic library UnsatisfiedLinkError - if the library does not exist.</p>	<pre>public void loadLibrary(String libName)</pre> <p>Since: API Level 1</p> <p>Loads and links the library with the specified name. The mapping of the specified library name to the full path for loading the library is implementation-dependent.</p> <p>Parameters</p> <p><i>libName</i> the name of the library to load.</p> <p>Throws</p> <p>UnsatisfiedLinkError if the library can not be loaded.</p> <p>SecurityException if the current <code>SecurityManager</code> does not allow to load the library.</p> <p>See Also</p> <p>checkLink(String)</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
NullPointerException - if libname is null See Also: SecurityException , SecurityManager.checkLink(java.lang.String)	
maxMemory <pre>public long maxMemory()</pre> <p>Returns the maximum amount of memory that the Java virtual machine will attempt to use. If there is no inherent limit then the value Long.MAX_VALUE will be returned.</p> <p>Returns: the maximum amount of memory that the virtual machine will attempt to use, measured in bytes</p> <p>Since: 1.4</p>	<pre>public long maxMemory ()</pre> <p>Since: API Level 1</p> <p>Returns the maximum amount of memory that may be used by the virtual machine, or <code>Long.MAX_VALUE</code> if there is no such limit.</p> <p>Returns the maximum amount of memory that the virtual machine will try to allocate, measured in bytes.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>removeShutdownHook</p> <p>public boolean removeShutdownHook(Thread hook)</p> <p>De-registers a previously-registered virtual-machine shutdown hook.</p> <p>Parameters: hook - the hook to remove</p> <p>Returns: true if the specified hook had previously been registered and was successfully de-registered, false otherwise.</p> <p>Throws: IllegalStateException - If the virtual machine is already in the process of shutting down SecurityException - If a security manager is present and it denies RuntimePermission("shutdownHooks")</p> <p>Since: 1.3</p> <p>See Also: addShutdownHook(java.lang.Thread), exit(int)</p>	<p>public boolean removeShutdownHook (Thread hook)</p> <p>Since: API Level 1</p> <p>Unregisters a previously registered virtual machine shutdown hook.</p> <p>Parameters hook the shutdown hook to remove.</p> <p>Returns true if the hook has been removed successfully; false otherwise.</p> <p>Throws IllegalStateException if the virtual machine is already shutting down. SecurityException if a SecurityManager is registered and the calling code doesn't have the RuntimePermission("shutdownHooks").</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>runFinalization</p> <pre>public void runFinalization()</pre> <p>Runs the finalization methods of any objects pending finalization. Calling this method suggests that the Java virtual machine expend effort toward running the <code>finalize</code> methods of objects that have been found to be discarded but whose <code>finalize</code> methods have not yet been run. When control returns from the method call, the virtual machine has made a best effort to complete all outstanding finalizations.</p> <p>The virtual machine performs the finalization process automatically as needed, in a separate thread, if the <code>runFinalization</code> method is not invoked explicitly.</p> <p>The method System.runFinalization() is the conventional and convenient means of invoking this method.</p> <p>See Also: Object.finalize()</p>	<pre>public void runFinalization ()</pre> <p>Since: API Level 1</p> <p>Provides a hint to the virtual machine that it would be useful to attempt to perform any outstanding object finalization.</p>

<p>Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)</p>	<p>Android APIs (java.lang.Runtime)</p>
<p>runFinalizersOnExit</p> <p>@Deprecated public static void runFinalizersOnExit(boolean value)</p> <p>Deprecated. <i>This method is inherently unsafe. It may result in finalizers being called on live objects while other threads are concurrently manipulating those objects, resulting in erratic behavior or deadlock.</i></p> <p>Enable or disable finalization on exit; doing so specifies that the finalizers of all objects that have finalizers that have not yet been automatically invoked are to be run before the Java runtime exits. By default, finalization on exit is disabled.</p> <p>If there is a security manager, its <code>checkExit</code> method is first called with 0 as its argument to ensure the exit is allowed. This could result in a <code>SecurityException</code>.</p> <p>Parameters: value - true to enable finalization on exit, false to disable</p> <p>Throws: SecurityException - if a security manager exists and its <code>checkExit</code> method doesn't allow the exit.</p> <p>Since: JDK1.1</p> <p>See Also: exit(int), gc(), SecurityManager.checkExit(int)</p>	<p>public static void runFinalizersOnExit (boolean run)</p> <p>Since: API Level 1</p> <p>This method is deprecated. This method is unsafe.</p> <p>Sets the flag that indicates whether all objects are finalized when the virtual machine is about to exit. Note that all finalization which occurs when the system is exiting is performed after all running threads have been terminated.</p> <p>Parameters run true to enable finalization on exit, false to disable it.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>totalMemory</p> <pre>public long totalMemory()</pre> <p>Returns the total amount of memory in the Java virtual machine. The value returned by this method may vary over time, depending on the host environment.</p> <p>Note that the amount of memory required to hold an object of any given type may be implementation-dependent.</p> <p>Returns: the total amount of memory currently available for current and future objects, measured in bytes.</p>	<pre>public long totalMemory ()</pre> <p>Since: API Level 1</p> <p>Returns the total amount of memory which is available to the running program.</p> <p>Returns the total amount of memory, measured in bytes.</p>
<p>traceInstructions</p> <pre>public void traceInstructions(boolean on)</pre> <p>Enables/Disables tracing of instructions. If the <code>boolean</code> argument is <code>true</code>, this method suggests that the Java virtual machine emit debugging information for each instruction in the virtual machine as it is executed. The format of this information, and the file or other output stream to which it is emitted, depends on the host environment. The virtual machine may ignore this request if it does not support this feature. The destination of the trace output is system dependent.</p> <p>If the <code>boolean</code> argument is <code>false</code>, this method causes the virtual machine to stop performing the detailed instruction trace it is performing.</p> <p>Parameters: <code>on</code> - <code>true</code> to enable instruction tracing; <code>false</code> to disable this feature.</p>	<pre>public void traceInstructions (boolean enable)</pre> <p>Since: API Level 1</p> <p>Switches the output of debug information for instructions on or off. On Android, this method does nothing.</p> <p>Parameters <code>enable</code> <code>true</code> to switch tracing on, <code>false</code> to switch it off.</p>

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
<p>traceMethodCalls</p> <pre>public void traceMethodCalls(boolean on)</pre> <p>Enables/Disables tracing of method calls. If the <code>boolean</code> argument is <code>true</code>, this method suggests that the Java virtual machine emit debugging information for each method in the virtual machine as it is called. The format of this information, and the file or other output stream to which it is emitted, depends on the host environment. The virtual machine may ignore this request if it does not support this feature.</p> <p>Calling this method with argument <code>false</code> suggests that the virtual machine cease emitting per-call debugging information.</p> <p>Parameters:</p> <p><code>on</code> - <code>true</code> to enable instruction tracing; <code>false</code> to disable this feature.</p>	<pre>public void traceMethodCalls (boolean enable)</pre> <p>Since: API Level 1</p> <p>Switches the output of debug information for methods on or off.</p> <p>Parameters</p> <p><code>enable</code> <code>true</code> to switch tracing on, <code>false</code> to switch it off.</p>

Exhibit Copyright-G

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security.ProtectionDomain)	Android Source Code ³ dalvik/libcore/security/src/main/java/java/security/ProtectionDomain.java
<div data-bbox="197 310 663 358" data-label="Section-Header"> <h3>Constructor Summary</h3> </div> <div data-bbox="197 375 890 505" data-label="Text"> <p>ProtectionDomain(CodeSource codesource, PermissionCollection permissions) Creates a new ProtectionDomain with the given CodeSource and Permissions.</p> </div>	<pre>public ProtectionDomain(CodeSource cs, PermissionCollection permissions) { this.codeSource = cs; if (permissions != null) { permissions.setReadOnly(); } this.permissions = permissions; //this.classLoader = null; //this.principals = null; //dynamicPerms = false; }</pre>
<div data-bbox="197 659 995 821" data-label="Text"> <p>ProtectionDomain(CodeSource codesource, PermissionCollection permissions, ClassLoader classloader, Principal[] principals) Creates a new ProtectionDomain qualified by the given CodeSource, Permissions, ClassLoader and array of Principals.</p> </div>	<pre>public ProtectionDomain(CodeSource cs, PermissionCollection permissions, ClassLoader cl, Principal[] principals) { this.codeSource = cs; if (permissions != null) { permissions.setReadOnly(); } this.permissions = permissions; this.classLoader = cl; if (principals != null) { this.principals = new Principal[principals.length]; System.arraycopy(principals, 0, this.principals, 0, this.principals.length); } dynamicPerms = true; }</pre>
<div data-bbox="197 1305 575 1354" data-label="Section-Header"> <h3>Method Summary</h3> </div> <div data-bbox="197 1370 890 1468" data-label="Text"> <p>ClassLoader getClassLoader() Returns the ClassLoader of this domain.</p> </div>	<pre>public final ClassLoader getClassLoader() { return classLoader; }</pre>

CodeSource	getCodeSource() Returns the CodeSource of this domain.	<pre>public final CodeSource getCodeSource() { return codeSource; }</pre>
PermissionCollection	getPermissions() Returns the static permissions granted to this domain.	<pre>public final PermissionCollection getPermissions() { return permissions; }</pre>
Principal[]	getPrincipals() Returns an array of principals for this domain.	<pre>public final Principal[] getPrincipals() { if(principals == null) { return new Principal[0]; } Principal[] tmp = new Principal[principals.length]; System.arraycopy(principals, 0, tmp, 0, tmp.length); return tmp; }</pre>
boolean	implies(Permission permission) Check and see if this ProtectionDomain implies the permissions expressed in the Permission object.	<pre>public boolean implies(Permission permission) { // First, test with the Policy, as the default Policy.implies() // checks for both dynamic and static collections of the // ProtectionDomain passed... if (dynamicPerms && Policy.getAccessiblePolicy().implies(this, permission)) { return true; } }</pre>
String	toString() Convert a ProtectionDomain to a String.	<pre>public String toString() { StringBuilder buf = new StringBuilder(200); buf.append("ProtectionDomain\n"); //\$NON- NLS-1\$ buf.append("CodeSource=").append(//\$NON- NLS-1\$ codeSource == null ? "<null>" : codeSource.toString()).append(//\$NON-NLS-1\$</pre>

```

        "\n"); //$NON-NLS-1$
        buf.append("ClassLoader=").append( //$NON-NLS-1$
            classLoader == null ? "<null>" :
            classLoader.toString()) //$NON-NLS-1$
            .append("\n"); //$NON-NLS-1$
        if (principals == null ||
            principals.length == 0) {
            buf.append("<no principals>\n");
        //$NON-NLS-1$
        } else {
            buf.append("Principals: <\n"); //$NON-NLS-1$
            for (int i = 0; i < principals.length;
                i++) {
                buf.append("\t").append( //$NON-NLS-1$
                    principals[i] == null ?
                    "<null>" : principals[i] //$NON-NLS-1$
                    .toString()).append("\n"); //$NON-NLS-1$
            }
            buf.append(">"); //$NON-NLS-1$
        }

        //permissions here
        buf.append("Permissions:\n"); //$NON-NLS-1$
        if (permissions == null) {
            buf.append("\t\t<no static
permissions>\n"); //$NON-NLS-1$
        } else {
            buf.append("\t\tstatic:
").append(permissions.toString()).append( //$NON-NLS-1$
                "\n"); //$NON-NLS-1$
        }

        if (dynamicPerms) {
            if (Policy.isSet()) {

```

	<pre> PermissionCollection perms; perms = Policy.getAccessiblePolicy().getPermissions(this); if (perms == null) { buf.append("\t\t<no dynamic permissions>\n"); //\$NON-NLS-1\$ } else { buf.append("\t\tdynamic: ").append(perms.toString()) //\$NON-NLS-1\$.append("\n"); //\$NON- NLS-1\$ } } else { buf.append("\t\t<no dynamic permissions>\n"); //\$NON-NLS-1\$ } } return buf.toString(); } </pre>
--	--

Exhibit Copyright-H

ProtectionDomain.java from Android 2.2 ("Froyo")

```

/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package java.security;

/**
 * {@code ProtectionDomain} represents all permissions that are granted to a
 * specific code source. The {@link ClassLoader} associates each class with the
 * corresponding {@code ProtectionDomain}, depending on the location and the
 * certificates (encapsulates in {@link CodeSource}) it loads the code from.
 * <p>
 * A class belongs to exactly one protection domain and the protection domain
 * can not be changed during the lifetime of the class.
 */
public class ProtectionDomain {

    // CodeSource for this ProtectionDomain
    private CodeSource codeSource;

    // Static permissions for this ProtectionDomain
    private PermissionCollection permissions;

    // ClassLoader
    private ClassLoader classLoader;

    // Set of principals associated with this ProtectionDomain
    private Principal[] principals;

    // false if this ProtectionDomain was constructed with static
    // permissions, true otherwise.
    private boolean dynamicPerms;

```

```

/**
 * Constructs a new instance of {@code ProtectionDomain} with the specified
 * code source and the specified static permissions.
 * <p>
 * If {@code permissions} is not {@code null}, the {@code permissions}
 * collection is made immutable by calling
 * {@link PermissionCollection#setReadOnly()} and it is considered as
 * granted statically to this {@code ProtectionDomain}.
 * <p>
 * The policy will not be consulted by access checks against this {@code
 * ProtectionDomain}.
 * <p>
 * If {@code permissions} is {@code null}, the method {@link
 * ProtectionDomain#implies(Permission)} always returns {@code false}.
 *
 * @param cs
 *         the code source associated with this domain, maybe {@code
 *         null}.
 * @param permissions
 *         the {@code PermissionCollection} containing all permissions to
 *         be statically granted to this {@code ProtectionDomain}, maybe
 *         {@code null}.
 */
public ProtectionDomain(CodeSource cs, PermissionCollection permissions) {
    this.codeSource = cs;
    if (permissions != null) {
        permissions.setReadOnly();
    }
    this.permissions = permissions;
    //this.classLoader = null;
    //this.principals = null;
    //dynamicPerms = false;
}

/**
 * Constructs a new instance of {@code ProtectionDomain} with the specified
 * code source, the permissions, the class loader and the principals.
 * <p>
 * If {@code permissions} is {@code null}, and access checks are performed
 * against this protection domain, the permissions defined by the policy are
 * consulted. If {@code permissions} is not {@code null}, the {@code
 * permissions} collection is made immutable by calling
 * {@link PermissionCollection#setReadOnly()}. If access checks are
 * performed, the policy and the provided permission collection are checked.
 * <p>
 * External modifications of the provided {@code principals} array has no
 * impact on this {@code ProtectionDomain}.
 *

```



```

* @param cs
*     the code source associated with this domain, maybe {@code
*     null}.
* @param permissions
*     the permissions associated with this domain, maybe {@code
*     null}.
* @param cl
*     the class loader associated with this domain, maybe {@code
*     null}.
* @param principals
*     the principals associated with this domain, maybe {@code
*     null}.
*/
public ProtectionDomain(CodeSource cs, PermissionCollection permissions,
    ClassLoader cl, Principal[] principals) {
    this.codeSource = cs;
    if (permissions != null) {
        permissions.setReadOnly();
    }
    this.permissions = permissions;
    this.classLoader = cl;
    if (principals != null) {
        this.principals = new Principal[principals.length];
        System.arraycopy(principals, 0, this.principals, 0,
            this.principals.length);
    }
    dynamicPerms = true;
}

/**
 * Returns the {@code ClassLoader} associated with this {@code
 * ProtectionDomain}.
 *
 * @return the {@code ClassLoader} associated with this {@code
 *     ProtectionDomain}, maybe {@code null}.
 */
public final ClassLoader getClassLoader() {
    return classLoader;
}

/**
 * Returns the {@code CodeSource} of this {@code ProtectionDomain}.
 *
 * @return the {@code CodeSource} of this {@code ProtectionDomain}, maybe
 *     {@code null}.
 */
public final CodeSource getCodeSource() {
    return codeSource;
}

```

```

/**
 * Returns the static permissions that are granted to this {@code
 * ProtectionDomain}.
 *
 * @return the static permissions that are granted to this {@code
 *         ProtectionDomain}, maybe {@code null}.
 */
public final PermissionCollection getPermissions() {
    return permissions;
}

/**
 * Returns the principals associated with this {@code ProtectionDomain}.
 * Modifications of the returned {@code Principal} array has no impact on
 * this {@code ProtectionDomain}.
 *
 * @return the principals associated with this {@code ProtectionDomain}.
 */
public final Principal[] getPrincipals() {
    if( principals == null ) {
        return new Principal[0];
    }
    Principal[] tmp = new Principal[principals.length];
    System.arraycopy(principals, 0, tmp, 0, tmp.length);
    return tmp;
}

/**
 * Indicates whether the specified permission is implied by this {@code
 * ProtectionDomain}.
 *
 * <p>
 * If this {@code ProtectionDomain} was constructed with
 * {@link #ProtectionDomain(CodeSource, PermissionCollection)}, the
 * specified permission is only checked against the permission collection
 * provided in the constructor. If {@code null} was provided, {@code false}
 * is returned.
 *
 * <p>
 * If this {@code ProtectionDomain} was constructed with
 * {@link #ProtectionDomain(CodeSource, PermissionCollection, ClassLoader, Principal[])}
 * , the specified permission is checked against the policy and the
 * permission collection provided in the constructor.
 *
 * @param permission
 *         the permission to check against the domain.
 * @return {@code true} if the specified {@code permission} is implied by
 *         this {@code ProtectionDomain}, {@code false} otherwise.
 */
public boolean implies(Permission permission) {

```

```

// First, test with the Policy, as the default Policy.implies()
// checks for both dynamic and static collections of the
// ProtectionDomain passed...
if (dynamicPerms
    && Policy.getAccessiblePolicy().implies(this, permission)) {
    return true;
}

// ... and we get here if
// either the permissions are static
// or Policy.implies() did not check for static permissions
// or the permission is not implied
return permissions == null ? false : permissions.implies(permission);
}

/**
 * Returns a string containing a concise, human-readable description of the
 * this {@code ProtectionDomain}.
 *
 * @return a printable representation for this {@code ProtectionDomain}.
 */
@Override
public String toString() {
    StringBuilder buf = new StringBuilder(200);
    buf.append("ProtectionDomain\n"); //$NON-NLS-1$
    buf.append("CodeSource=").append( //$NON-NLS-1$
        codeSource == null ? "<null>" : codeSource.toString()).append( //$NON-NLS-1$
        "\n"); //$NON-NLS-1$
    buf.append("ClassLoader=").append( //$NON-NLS-1$
        classLoader == null ? "<null>" : classLoader.toString()) //$NON-NLS-1$
        .append("\n"); //$NON-NLS-1$
    if (principals == null || principals.length == 0) {
        buf.append("<no principals>\n"); //$NON-NLS-1$
    } else {
        buf.append("Principals: <\n"); //$NON-NLS-1$
        for (int i = 0; i < principals.length; i++) {
            buf.append("\t").append( //$NON-NLS-1$
                principals[i] == null ? "<null>" : principals[i] //$NON-NLS-1$
                    .toString()).append("\n"); //$NON-NLS-1$
        }
        buf.append(">"); //$NON-NLS-1$
    }

    //permissions here
    buf.append("Permissions:\n"); //$NON-NLS-1$
    if (permissions == null) {
        buf.append("\t\t<no static permissions>\n"); //$NON-NLS-1$
    } else {
        buf.append("\t\tstatic: ").append(permissions.toString()).append( //$NON-NLS-1$

```

```

        "\n"); //$NON-NLS-1$
    }
    if (dynamicPerms) {
        if (Policy.isSet()) {
            PermissionCollection perms;
            perms = Policy.getAccessiblePolicy().getPermissions(this);
            if (perms == null) {
                buf.append("\t\t<no dynamic permissions>\n"); //$NON-NLS-1$
            } else {
                buf.append("\t\tdynamic: ").append(perms.toString()) //$NON-NLS-1$
                    .append("\n"); //$NON-NLS-1$
            }
        } else {
            buf.append("\t\t<no dynamic permissions>\n"); //$NON-NLS-1$
        }
    }
    return buf.toString();
}
}

```

Exhibit Copyright-I

readme.txt from SGH-I897_OpenSource.tar.gz

How to build

1. Get android open source.
: version info - Android eclair 2.1 (android-2.1_r2)
(Download site : <http://source.android.com>)
2. Overwrite modules that you want to build.
3. Add the following lines at the end of build/target/board/generic/BoardConfig.mk

```
BOARD_HAVE_BLUETOOTH := true
BT_USE_BTL_IF := true
BT_ALT_STACK := true
BRCM_BTL_INCLUDE_A2DP := true
BRCM_BT_USE_BTL_IF := true
```

4. make update-api
5. make

Exhibit Copyright-J

PolicyNodeImpl.jad (decompiled version of Oracle PolicyNodeImpl.class) [spacing adjusted for comparison]	PolicyNodeImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: PolicyNodeImpl.java package sun.security.provider.certpath; import java.security.cert.PolicyNode; import java.util.Collections; import java.util.HashSet; import java.util.Iterator; import java.util.Set; final class PolicyNodeImpl implements PolicyNode { private static final String ANY_POLICY = "2.5.29.32.0"; private PolicyNodeImpl mParent; private HashSet mChildren; private String mValidPolicy; private HashSet mQualifierSet; private boolean mCriticalityIndicator; private HashSet mExpectedPolicySet; private boolean mOriginalExpectedPolicySet; private int mDepth; private boolean isImmutable; PolicyNodeImpl(PolicyNodeImpl policynodeimpl, String s, Set set, boolean flag, Set set1, boolean flag1) { isImmutable = false; mParent = policynodeimpl; mChildren = new HashSet(); if(s != null) mValidPolicy = s; else mValidPolicy = ""; if(set != null) mQualifierSet = new HashSet(set); else mQualifierSet = new HashSet(); mCriticalityIndicator = flag; </pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.cert; import java.security.cert.PolicyNode; import java.util.*; public class PolicyNodeImpl implements PolicyNode { private static final String ANY_POLICY = "2.5.29.32.0"; private PolicyNodeImpl mParent; private HashSet mChildren; private String mValidPolicy; private HashSet mQualifierSet; private boolean mCriticalityIndicator; private HashSet mExpectedPolicySet; private boolean mOriginalExpectedPolicySet; private int mDepth; private boolean isImmutable; public PolicyNodeImpl(PolicyNodeImpl policynodeimpl, String s, Set set, boolean flag, Set set1, boolean flag1) { isImmutable = false; mParent = policynodeimpl; mChildren = new HashSet(); if(s != null) { mValidPolicy = s; } else { mValidPolicy = ""; } if(set != null) { mQualifierSet = new HashSet(set); } else { mQualifierSet = new HashSet(); } mCriticalityIndicator = flag; </pre>

PolicyNodeImpl.jad (decompiled version of Oracle PolicyNodeImpl.class) [spacing adjusted for comparison]	PolicyNodeImpl.java (Android version) [spacing adjusted for comparison]
<pre> if(set1 != null) mExpectedPolicySet = new HashSet(set1); else mExpectedPolicySet = new HashSet(); mOriginalExpectedPolicySet = !flag1; if(mParent != null) { mDepth = mParent.getDepth() + 1; mParent.addChild(this); } else { mDepth = 0; } } PolicyNodeImpl(PolicyNodeImpl polycynodeimpl, PolicyNodeImpl polycynodeimpl1) { this(polycynodeimpl, polycynodeimpl1.mValidPolicy, ((Set) (polycynodeimpl1.mQualifierSet)), polycynodeimpl1.mCriticalityIndicator, ((Set) (polycynodeimpl1.mExpectedPolicySet)), false); } public PolicyNode getParent() { return mParent; } public Iterator getChildren() { return Collections.unmodifiableSet(mChildren).iterator(); } public int getDepth() { return mDepth; } public String getValidPolicy() { return mValidPolicy; } public Set getPolicyQualifiers() { return Collections.unmodifiableSet(mQualifierSet); } public Set getExpectedPolicies() { return Collections.unmodifiableSet(mExpectedPolicySet); } public boolean isCritical() { return mCriticalityIndicator; } public String toString() { StringBuffer stringbuffer = new StringBuffer(asString()); for(Iterator iterator = getChildren(); iterator.hasNext(); stringbuffer.append((PolicyNodeImpl)iterator.next())); return stringbuffer.toString(); } boolean isImmutable() { return isImmutable; } </pre>	<pre> if(set1 != null) { mExpectedPolicySet = new HashSet(set1); } else { mExpectedPolicySet = new HashSet(); } mOriginalExpectedPolicySet = !flag1; if(mParent != null) { mDepth = mParent.getDepth() + 1; mParent.addChild(this); } else { mDepth = 0; } } PolicyNodeImpl(PolicyNodeImpl polycynodeimpl, PolicyNodeImpl polycynodeimpl1) { this(polycynodeimpl, polycynodeimpl1.mValidPolicy, ((Set) (polycynodeimpl1.mQualifierSet)), polycynodeimpl1.mCriticalityIndicator, ((Set) (polycynodeimpl1.mExpectedPolicySet)), false); } public PolicyNode getParent() { return mParent; } public Iterator getChildren() { return Collections.unmodifiableSet(mChildren).iterator(); } public int getDepth() { return mDepth; } public String getValidPolicy() { return mValidPolicy; } public Set getPolicyQualifiers() { return Collections.unmodifiableSet(mQualifierSet); } public Set getExpectedPolicies() { return Collections.unmodifiableSet(mExpectedPolicySet); } public boolean isCritical() { return mCriticalityIndicator; } public String toString() { StringBuffer stringbuffer = new StringBuffer(asString()); for(Iterator iterator = getChildren(); iterator.hasNext(); stringbuffer.append((PolicyNodeImpl)iterator.next())); return stringbuffer.toString(); } boolean isImmutable() { return isImmutable; } </pre>

PolicyNodeImpl.jad (decompiled version of Oracle PolicyNodeImpl.class) [spacing adjusted for comparison]	PolicyNodeImpl.java (Android version) [spacing adjusted for comparison]
<pre> void setImmutable() { if(!immutable) return; PolicyNodeImpl polycnodeimpl; for(Iterator iterator = mChildren.iterator(); iterator.hasNext(); polycnodeimpl.setImmutable()) polycnodeimpl = (PolicyNodeImpl)iterator.next(); immutable = true; } private void addChild(PolicyNodeImpl polycnodeimpl) { if(!immutable) { throw new IllegalStateException("PolicyNode is immutable"); } else { mChildren.add(polycnodeimpl); return; } } void addExpectedPolicy(String s) { if(!immutable) throw new IllegalStateException("PolicyNode is immutable"); if(mOriginalExpectedPolicySet) { mExpectedPolicySet.clear(); mOriginalExpectedPolicySet = false; } mExpectedPolicySet.add(s); } void prune(int i) { if(!immutable) throw new IllegalStateException("PolicyNode is immutable"); if(mChildren.size() == 0) return; Iterator iterator = mChildren.iterator(); do { if(!iterator.hasNext()) break; PolicyNodeImpl polycnodeimpl = (PolicyNodeImpl)iterator.next(); polycnodeimpl.prune(i); if(polycnodeimpl.mChildren.size() == 0 && i > mDepth + 1) iterator.remove(); } while(true); } void deleteChild(PolicyNode polycnode) { if(!immutable) { throw new IllegalStateException("PolicyNode is immutable"); } else { mChildren.remove(polycnode); return; } } PolicyNodeImpl copyTree() { return copyTree(null); } </pre>	<pre> void setImmutable() { if(!immutable) return; PolicyNodeImpl polycnodeimpl; for(Iterator iterator = mChildren.iterator(); iterator.hasNext(); polycnodeimpl.setImmutable()) polycnodeimpl = (PolicyNodeImpl)iterator.next(); immutable = true; } private void addChild(PolicyNodeImpl polycnodeimpl) { if(!immutable) { throw new IllegalStateException("PolicyNode is immutable"); } else { mChildren.add(polycnodeimpl); return; } } void addExpectedPolicy(String s) { if(!immutable) throw new IllegalStateException("PolicyNode is immutable"); if(mOriginalExpectedPolicySet) { mExpectedPolicySet.clear(); mOriginalExpectedPolicySet = false; } mExpectedPolicySet.add(s); } void prune(int i) { if(!immutable) throw new IllegalStateException("PolicyNode is immutable"); if(mChildren.size() == 0) return; Iterator iterator = mChildren.iterator(); do { if(!iterator.hasNext()) break; PolicyNodeImpl polycnodeimpl = (PolicyNodeImpl)iterator.next(); polycnodeimpl.prune(i); if(polycnodeimpl.mChildren.size() == 0 && i > mDepth + 1) iterator.remove(); } while(true); } void deleteChild(PolicyNode polycnode) { if(!immutable) { throw new IllegalStateException("PolicyNode is immutable"); } else { mChildren.remove(polycnode); return; } } PolicyNodeImpl copyTree() { return copyTree(null); } </pre>

PolicyNodeImpl.jad (decompiled version of Oracle PolicyNodeImpl.class) [spacing adjusted for comparison]	PolicyNodeImpl.java (Android version) [spacing adjusted for comparison]
<pre> private PolicyNodeImpl copyTree(PolicyNodeImpl pol i cynodei mpl) { PolicyNodeImpl pol i cynodei mpl 1 = new PolicyNodeImpl (pol i cynodei mpl , thi s); PolicyNodeImpl pol i cynodei mpl 2; for(Iterator i terator = mChildren.i terator(); i terator.hasNext(); pol i cynodei mpl 2.copyTree(pol i cynodei mpl 1)) pol i cynodei mpl 2 = (PolicyNodeImpl)i terator.next(); return pol i cynodei mpl 1; } Set getPolicyNodes(int i) { HashSet hashset = new HashSet(); getPolicyNodes(i, ((Set) (hashset))); return hashset; } private void getPolicyNodes(int i, Set set) { if(mDepth == i) { set.add(thi s); } el se { PolicyNodeImpl pol i cynodei mpl ; for(Iterator i terator = mChildren.i terator(); i terator.hasNext(); pol i cynodei mpl .getPol i cyNodes(i, set)) pol i cynodei mpl = (Pol i cyNodeI mpl)i terator.next(); } } Set getPolicyNodesExpected(int i, String s, boolean flag) { if(s.equals("2.5.29.32.0")) return getPolicyNodes(i); el se return getPolicyNodesExpectedHel per(i, s, flag); } private Set getPolicyNodesExpectedHel per(int i, String s, boolean flag) { HashSet hashset = new HashSet(); if(mDepth < i) { PolicyNodeImpl pol i cynodei mpl ; for(Iterator i terator = mChildren.i terator(); i terator.hasNext(); hashset.addAll(pol i cynodei mpl .getPol i cyNodesExpectedHel per(i, s, flag))) pol i cynodei mpl = (Pol i cyNodeI mpl)i terator.next(); } el se if(flag) { if(mExpectedPol i cySet.contains("2.5.29.32.0")) hashset.add(thi s); } el se if(mExpectedPol i cySet.contains(s)) hashset.add(thi s); return hashset; } Set getPolicyNodesValid(int i, String s) { HashSet hashset = new HashSet(); </pre>	<pre> private PolicyNodeImpl copyTree(PolicyNodeImpl pol i cynodei mpl) { PolicyNodeImpl pol i cynodei mpl 1 = new PolicyNodeImpl (pol i cynodei mpl , thi s); PolicyNodeImpl pol i cynodei mpl 2; for(Iterator i terator = mChildren.i terator(); i terator.hasNext(); pol i cynodei mpl 2.copyTree(pol i cynodei mpl 1)) pol i cynodei mpl 2 = (Pol i cyNodeI mpl)i terator.next(); return pol i cynodei mpl 1; } Set getPolicyNodes(int i) { HashSet hashset = new HashSet(); getPolicyNodes(i, ((Set) (hashset))); return hashset; } private void getPolicyNodes(int i, Set set) { if(mDepth == i) { set.add(thi s); } el se { PolicyNodeImpl pol i cynodei mpl ; for(Iterator i terator = mChildren.i terator(); i terator.hasNext(); pol i cynodei mpl .getPol i cyNodes(i, set)) pol i cynodei mpl = (Pol i cyNodeI mpl)i terator.next(); } } Set getPolicyNodesExpected(int i, String s, boolean flag) { if(s.equals("2.5.29.32.0")) return getPolicyNodes(i); el se return getPolicyNodesExpectedHel per(i, s, flag); } private Set getPolicyNodesExpectedHel per(int i, String s, boolean flag) { HashSet hashset = new HashSet(); if(mDepth < i) { PolicyNodeImpl pol i cynodei mpl ; for(Iterator i terator = mChildren.i terator(); i terator.hasNext(); hashset.addAll(pol i cynodei mpl .getPol i cyNodesExpectedHel per(i, s, flag))) pol i cynodei mpl = (Pol i cyNodeI mpl)i terator.next(); } el se if(flag) { if(mExpectedPol i cySet.contains("2.5.29.32.0")) hashset.add(thi s); } el se if(mExpectedPol i cySet.contains(s)) { hashset.add(thi s); } return hashset; } Set getPolicyNodesValid(int i, String s) { HashSet hashset = new HashSet(); </pre>

PolicyNodeImpl.jad (decompiled version of Oracle PolicyNodeImpl.class) [spacing adjusted for comparison]	PolicyNodeImpl.java (Android version) [spacing adjusted for comparison]
<pre> if(mDepth < i) { PolicyNodeImpl polycnodeimpl; for(Iterator iterator = mChildren.iterator(); iterator.hasNext(); hashset.addAll(polycnodeimpl.getPolicyNodesValid(i, s))) polycnodeimpl = (PolicyNodeImpl)iterator.next(); } else if(mValidPolicy.equals(s)) hashset.add(this); return hashset; } private static String policyToString(String s) { if(s.equals("2.5.29.32.0")) return "anyPolicy"; else return s; } String asString() { if(mParent == null) return "anyPolicy ROOT\n"; StringBuffer stringbuffer = new StringBuffer(); int i = 0; for(int j = getDepth(); i < j; i++) stringbuffer.append(" "); stringbuffer.append(policyToString(getValidPolicy())); stringbuffer.append(" CRIT: "); stringbuffer.append(isCritical()); stringbuffer.append(" EP: "); for(Iterator iterator = getExpectedPolicies().iterator(); iterator.hasNext(); stringbuffer.append(" ")) { String s = (String)iterator.next(); stringbuffer.append(policyToString(s)); } stringbuffer.append(" ("); stringbuffer.append(getDepth()); stringbuffer.append(")\n"); return stringbuffer.toString(); } } </pre>	<pre> if(mDepth < i) { PolicyNodeImpl polycnodeimpl; for(Iterator iterator = mChildren.iterator(); iterator.hasNext(); hashset.addAll(polycnodeimpl.getPolicyNodesValid(i, s))) polycnodeimpl = (PolicyNodeImpl)iterator.next(); } else if(mValidPolicy.equals(s)) { hashset.add(this); } return hashset; } private static String policyToString(String s) { if(s.equals("2.5.29.32.0")) { return "anyPolicy"; } else { return s; } } String asString() { if(mParent == null) return "anyPolicy ROOT\n"; StringBuffer stringbuffer = new StringBuffer(); int i = 0; for(int j = getDepth(); i < j; i++) stringbuffer.append(" "); stringbuffer.append(policyToString(getValidPolicy())); stringbuffer.append(" CRIT: "); stringbuffer.append(isCritical()); stringbuffer.append(" EP: "); for(Iterator iterator = getExpectedPolicies().iterator(); iterator.hasNext(); stringbuffer.append(" ")) { String s = (String)iterator.next(); stringbuffer.append(policyToString(s)); } stringbuffer.append(" ("); stringbuffer.append(getDepth()); stringbuffer.append(")\n"); return stringbuffer.toString(); } } </pre>

Exhibit Copyright-K

AclEntryImpl.jad (decompiled version of Oracle AclEntryImpl.class) [spacing adjusted for comparison]	AclEntryImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: AclEntryImpl.java package sun.security.acl; import java.security.Principal; import java.security.acl.AclEntry; import java.security.acl.Group; import java.security.acl.Permission; import java.util.Enumeration; import java.util.Vector; public class AclEntryImpl implements AclEntry { private Principal user; private Vector permissionSet; private boolean negative; public AclEntryImpl(Principal principal) { user = null; permissionSet = new Vector(10, 10); negative = false; user = principal; } public AclEntryImpl() { user = null; permissionSet = new Vector(10, 10); negative = false; } public boolean setPrincipal(Principal principal) { if(user != null) { return false; } else { user = principal; return true; } } public void setNegativePermissions() {</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.Principal; import java.security.acl.*; import java.util.Enumeration; import java.util.Vector; /** * Additional class for verification AclEntry interface */ public class AclEntryImpl implements AclEntry { private Principal user; private Vector permissionSet; private boolean negative; public AclEntryImpl(Principal principal) { user = null; permissionSet = new Vector(10, 10); negative = false; user = principal; } public AclEntryImpl() { user = null; permissionSet = new Vector(10, 10); negative = false; } public boolean setPrincipal(Principal principal) { if(user != null) { return false; } else { user = principal; return true; } } public void setNegativePermissions() {</pre>

AclEntryImpl.jad (decompiled version of Oracle AclEntryImpl.class) [spacing adjusted for comparison]	AclEntryImpl.java (Android version) [spacing adjusted for comparison]
<pre> negative = true; } public boolean isNegative() { return negative; } public boolean addPermission(Permission permission) { if(permissionSet.contains(permission)) { return false; } else { permissionSet.addElement(permission); return true; } } public boolean removePermission(Permission permission) { return permissionSet.removeElement(permission); } public boolean checkPermission(Permission permission) { return permissionSet.contains(permission); } public Enumeration permissions() { return permissionSet.elements(); } public String toString() { StringBuffer stringbuffer = new StringBuffer(); if(negative) stringbuffer.append("-"); else stringbuffer.append("+"); if(user instanceof Group) stringbuffer.append("Group. "); else stringbuffer.append("User. "); stringbuffer.append((new StringBui lder()).append(user).append("=").toString()); Enumeration enumeration = permissions(); do { if(!enumeration.hasMoreElements()) break; Permission permission = (Permission)enumeration.nextElement(); stringbuffer.append(permission); if(enumeration.hasMoreElements()) stringbuffer.append(","); } while(true); return new String(stringbuffer); } public synchronized Object clone() { AclEntryImpl aclentryimpl = new AclEntryImpl(user); aclentryimpl.permissionSet = (Vector)permissionSet.clone(); aclentryimpl.negative = negative; return aclentryimpl; } </pre>	<pre> negative = true; } public boolean isNegative() { return negative; } public boolean addPermission(Permission permission) { if(permissionSet.contains(permission)) { return false; } else { permissionSet.addElement(permission); return true; } } public boolean removePermission(Permission permission) { return permissionSet.removeElement(permission); } public boolean checkPermission(Permission permission) { return permissionSet.contains(permission); } public Enumeration permissions() { return permissionSet.elements(); } public String toString() { StringBuffer stringbuffer = new StringBuffer(); if(negative) stringbuffer.append("-"); else stringbuffer.append("+"); if(user instanceof Group) stringbuffer.append("Group. "); else stringbuffer.append("User. "); stringbuffer.append((new StringBui lder()).append(user).append("=").toString()); Enumeration enumeration = permissions(); do { if(!enumeration.hasMoreElements()) break; Permission permission = (Permission)enumeration.nextElement(); stringbuffer.append(permission); if(enumeration.hasMoreElements()) stringbuffer.append(","); } while(true); return new String(stringbuffer); } public synchronized Object clone() { AclEntryImpl aclentryimpl = new AclEntryImpl(user); aclentryimpl.permissionSet = (Vector)permissionSet.clone(); aclentryimpl.negative = negative; return aclentryimpl; } </pre>

AclEntryImpl.jad (decompiled version of Oracle AclEntryImpl.class) [spacing adjusted for comparison]	AclEntryImpl.java (Android version) [spacing adjusted for comparison]
<pre> public Principal getPrincipal () { return user; } </pre>	<pre> public Principal getPrincipal () { return user; } </pre>

Exhibit Copyright-L

AclImpl.jad (decompiled version of Oracle AclImpl.class) [spacing adjusted for comparison]	AclImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: AclImpl.java package sun.security.acl; import java.security.Principal; import java.security.acl.Acl; import java.security.acl.AclEntry; import java.security.acl.Group; import java.security.acl.NotOwnerException; import java.security.acl.Permission; import java.util.Enumeration; import java.util.Hashtable; import java.util.Vector; // Referenced classes of package sun.security.acl: // OwnerImpl, AclEnumerator public class AclImpl extends OwnerImpl implements Acl { private Hashtable allowedUsersTable; private Hashtable allowedGroupsTable; private Hashtable deniedUsersTable; private Hashtable deniedGroupsTable; private String aclName; private Vector zeroSet; public AclImpl(Principal principal, String s) { super(principal); allowedUsersTable = new Hashtable(23); allowedGroupsTable = new Hashtable(23); deniedUsersTable = new Hashtable(23); deniedGroupsTable = new Hashtable(23); aclName = null; zeroSet = new Vector(1, 1); try { setName(principal, s); } catch(Exception exception) { } } }</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.Principal; import java.security.acl.*; import java.util.*; /** * Additional class for verification Acl interface */ public class AclImpl extends OwnerImpl implements Acl { private Hashtable allowedUsersTable; private Hashtable allowedGroupsTable; private Hashtable deniedUsersTable; private Hashtable deniedGroupsTable; private String aclName; private Vector zeroSet; public AclImpl(Principal principal, String s) { super(principal); allowedUsersTable = new Hashtable(23); allowedGroupsTable = new Hashtable(23); deniedUsersTable = new Hashtable(23); deniedGroupsTable = new Hashtable(23); aclName = null; zeroSet = new Vector(1, 1); try { setName(principal, s); } catch(Exception exception) { } } }</pre>

AclImpl.jad (decompiled version of Oracle AclImpl.class) [spacing adjusted for comparison]	AclImpl.java (Android version) [spacing adjusted for comparison]
<pre> public void setName(Principal principal, String s) throws NotOwnerException { if(!isOwner(principal)) { throw new NotOwnerException(); } else { aclName = s; return; } } public String getName() { return aclName; } public synchronized boolean addEntry(Principal principal, AclEntry aclEntry) throws NotOwnerException { if(!isOwner(principal)) throw new NotOwnerException(); Hashtable hashtable = findTable(aclEntry); Principal principal1 = aclEntry.getPrincipal(); if(hashtable.get(principal1) != null) { return false; } else { hashtable.put(principal1, aclEntry); return true; } } public synchronized boolean removeEntry(Principal principal, AclEntry aclEntry) throws NotOwnerException { if(!isOwner(principal)) { throw new NotOwnerException(); } else { Hashtable hashtable = findTable(aclEntry); Principal principal1 = aclEntry.getPrincipal(); Object obj = hashtable.remove(principal1); return obj != null; } } public synchronized Enumeration getPermissions(Principal principal) { Enumeration enumeration2 = subtract(getGroupPositive(principal), getGroupNegative(principal)); Enumeration enumeration3 = subtract(getGroupNegative(principal), getGroupPositive(principal)); Enumeration enumeration = subtract(getIndividualPositive(principal), getIndividualNegative(principal)); Enumeration enumeration1 = subtract(getIndividualNegative(principal), getIndividualPositive(principal)); Enumeration enumeration4 = subtract(enumeration2, enumeration1); Enumeration enumeration5 = union(enumeration, enumeration4); enumeration = subtract(getIndividualPositive(principal), getIndividualNegative(principal)); enumeration1 = subtract(getIndividualNegative(principal), getIndividualPositive(principal)); enumeration4 = subtract(enumeration3, enumeration); Enumeration enumeration6 = union(enumeration1, enumeration4); </pre>	<pre> public void setName(Principal principal, String s) throws NotOwnerException { if(!isOwner(principal)) { throw new NotOwnerException(); } else { aclName = s; return; } } public String getName() { return aclName; } public synchronized boolean addEntry(Principal principal, AclEntry aclEntry) throws NotOwnerException { if(!isOwner(principal)) throw new NotOwnerException(); Hashtable hashtable = findTable(aclEntry); Principal principal1 = aclEntry.getPrincipal(); if(hashtable.get(principal1) != null) { return false; } else { hashtable.put(principal1, aclEntry); return true; } } public synchronized boolean removeEntry(Principal principal, AclEntry aclEntry) throws NotOwnerException { if(!isOwner(principal)) { throw new NotOwnerException(); } else { Hashtable hashtable = findTable(aclEntry); Principal principal1 = aclEntry.getPrincipal(); Object obj = hashtable.remove(principal1); return obj != null; } } public synchronized Enumeration getPermissions(Principal principal) { Enumeration enumeration2 = subtract(getGroupPositive(principal), getGroupNegative(principal)); Enumeration enumeration3 = subtract(getGroupNegative(principal), getGroupPositive(principal)); Enumeration enumeration = subtract(getIndividualPositive(principal), getIndividualNegative(principal)); Enumeration enumeration1 = subtract(getIndividualNegative(principal), getIndividualPositive(principal)); Enumeration enumeration4 = subtract(enumeration2, enumeration1); Enumeration enumeration5 = union(enumeration, enumeration4); enumeration = subtract(getIndividualPositive(principal), getIndividualNegative(principal)); enumeration1 = subtract(getIndividualNegative(principal), getIndividualPositive(principal)); enumeration4 = subtract(enumeration3, enumeration); Enumeration enumeration6 = union(enumeration1, enumeration4); </pre>

AclImpl.jad (decompiled version of Oracle AclImpl.class) [spacing adjusted for comparison]	AclImpl.java (Android version) [spacing adjusted for comparison]
<pre> return subtract(enumeration5, enumeration6); } public boolean checkPermission(Principal principal, Permission permission) { for(Enumeration enumeration = getPermissions(principal); enumeration.hasMoreElements();) { Permission permission1 = (Permission)enumeration.nextElement(); if(permission1.equals(permission)) return true; } return false; } public synchronized Enumeration entries() { return new AclEnumerator(this, allowedUsersTable, allowedGroupsTable, deniedUsersTable, deniedGroupsTable); } public String toString() { StringBuffer stringbuffer = new StringBuffer(); for(Enumeration enumeration = entries(); enumeration.hasMoreElements(); stringbuffer.append("\n")) { AclEntry aclentry = (AclEntry)enumeration.nextElement(); stringbuffer.append(aclentry.toString().trim()); } return stringbuffer.toString(); } private Hashtable findTable(AclEntry aclentry) { Hashtable hashtable = null; Principal principal = aclentry.getPrincipal(); if(principal instanceof Group) { if(aclentry.isNegative()) hashtable = deniedGroupsTable; else hashtable = allowedGroupsTable; } else if(aclentry.isNegative()) hashtable = deniedUsersTable; else hashtable = allowedUsersTable; return hashtable; } private static Enumeration union(Enumeration enumeration, Enumeration enumeration1) { Vector vector = new Vector(20, 20); for(; enumeration.hasMoreElements(); vector.addElement(enumeration.nextElement())); do { if(!enumeration1.hasMoreElements()) break; Object obj = enumeration1.nextElement(); if(!vector.contains(obj)) vector.addElement(obj); } while(true); } </pre>	<pre> return subtract(enumeration5, enumeration6); } public boolean checkPermission(Principal principal, Permission permission) { for(Enumeration enumeration = getPermissions(principal); enumeration.hasMoreElements();) { Permission permission1 = (Permission)enumeration.nextElement(); if(permission1.equals(permission)) return true; } return false; } public synchronized Enumeration entries() { return new AclEnumerator(this, allowedUsersTable, allowedGroupsTable, deniedUsersTable, deniedGroupsTable); } public String toString() { StringBuffer stringbuffer = new StringBuffer(); for(Enumeration enumeration = entries(); enumeration.hasMoreElements(); stringbuffer.append("\n")) { AclEntry aclentry = (AclEntry)enumeration.nextElement(); stringbuffer.append(aclentry.toString().trim()); } return stringbuffer.toString(); } private Hashtable findTable(AclEntry aclentry) { Hashtable hashtable = null; Principal principal = aclentry.getPrincipal(); if(principal instanceof Group) { if(aclentry.isNegative()) hashtable = deniedGroupsTable; else hashtable = allowedGroupsTable; } else if(aclentry.isNegative()) hashtable = deniedUsersTable; else hashtable = allowedUsersTable; return hashtable; } private static Enumeration union(Enumeration enumeration, Enumeration enumeration1) { Vector vector = new Vector(20, 20); for(; enumeration.hasMoreElements(); vector.addElement(enumeration.nextElement())); do { if(!enumeration1.hasMoreElements()) break; Object obj = enumeration1.nextElement(); if(!vector.contains(obj)) vector.addElement(obj); } while(true); } </pre>

AclImpl.jad (decompiled version of Oracle AclImpl.class) [spacing adjusted for comparison]	AclImpl.java (Android version) [spacing adjusted for comparison]
<pre> return vector.elements(); } private Enumeration subtract(Enumeration enumeration, Enumeration enumeration1) { Vector vector = new Vector(20, 20); for(; enumeration.hasMoreElements(); vector.addElement(enumeration.nextElement())); do { if(!enumeration1.hasMoreElements()) break; Object obj = enumeration1.nextElement(); if(vector.contains(obj)) vector.removeElement(obj); } while(true); return vector.elements(); } private Enumeration getGroupPositive(Principal principal) { Enumeration enumeration = zeroSet.elements(); Enumeration enumeration1 = allowedGroupsTable.keys(); do { if(!enumeration1.hasMoreElements()) break; Group group = (Group)enumeration1.nextElement(); if(group.isMember(principal)) { AclEntry aclEntry = (AclEntry)allowedGroupsTable.get(group); enumeration = union(aclEntry.permissions(), enumeration); } } while(true); return enumeration; } private Enumeration getGroupNegative(Principal principal) { Enumeration enumeration = zeroSet.elements(); Enumeration enumeration1 = deniedGroupsTable.keys(); do { if(!enumeration1.hasMoreElements()) break; Group group = (Group)enumeration1.nextElement(); if(group.isMember(principal)) { AclEntry aclEntry = (AclEntry)deniedGroupsTable.get(group); enumeration = union(aclEntry.permissions(), enumeration); } } while(true); return enumeration; } private Enumeration getIndividualPositive(Principal principal) { Enumeration enumeration = zeroSet.elements(); AclEntry aclEntry = (AclEntry)allowedUsersTable.get(principal); if(aclEntry != null) enumeration = aclEntry.permissions(); return enumeration; } private Enumeration getIndividualNegative(Principal principal) { Enumeration enumeration = zeroSet.elements(); AclEntry aclEntry = (AclEntry)deniedUsersTable.get(principal); </pre>	<pre> return vector.elements(); } private Enumeration subtract(Enumeration enumeration, Enumeration enumeration1) { Vector vector = new Vector(20, 20); for(; enumeration.hasMoreElements(); vector.addElement(enumeration.nextElement())); do { if(!enumeration1.hasMoreElements()) break; Object obj = enumeration1.nextElement(); if(vector.contains(obj)) vector.removeElement(obj); } while(true); return vector.elements(); } private Enumeration getGroupPositive(Principal principal) { Enumeration enumeration = zeroSet.elements(); Enumeration enumeration1 = allowedGroupsTable.keys(); do { if(!enumeration1.hasMoreElements()) break; Group group = (Group)enumeration1.nextElement(); if(group.isMember(principal)) { AclEntry aclEntry = (AclEntry)allowedGroupsTable.get(group); enumeration = union(aclEntry.permissions(), enumeration); } } while(true); return enumeration; } private Enumeration getGroupNegative(Principal principal) { Enumeration enumeration = zeroSet.elements(); Enumeration enumeration1 = deniedGroupsTable.keys(); do { if(!enumeration1.hasMoreElements()) break; Group group = (Group)enumeration1.nextElement(); if(group.isMember(principal)) { AclEntry aclEntry = (AclEntry)deniedGroupsTable.get(group); enumeration = union(aclEntry.permissions(), enumeration); } } while(true); return enumeration; } private Enumeration getIndividualPositive(Principal principal) { Enumeration enumeration = zeroSet.elements(); AclEntry aclEntry = (AclEntry)allowedUsersTable.get(principal); if(aclEntry != null) enumeration = aclEntry.permissions(); return enumeration; } private Enumeration getIndividualNegative(Principal principal) { Enumeration enumeration = zeroSet.elements(); AclEntry aclEntry = (AclEntry)deniedUsersTable.get(principal); </pre>

AclImpl.jad (decompiled version of Oracle AclImpl.class) [spacing adjusted for comparison]	AclImpl.java (Android version) [spacing adjusted for comparison]
<pre> if(acl entry != null) enumerati on = acl entry. permi ssi ons(); return enumerati on; } } </pre>	<pre> if(acl entry != null) enumerati on = acl entry. permi ssi ons(); return enumerati on; } } </pre>

Exhibit Copyright-M

GroupImpl.jad (decompiled version of Oralce GroupImpl.class) [spacing adjusted for comparison]	GroupImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: GroupImpl.java package sun.security.acl; import java.security.Principal; import java.security.acl.Group; import java.util.Enumeration; import java.util.Vector; public class GroupImpl implements Group { private Vector groupMembers; private String group; public GroupImpl(String s) { groupMembers = new Vector(50, 100); group = s; } public boolean addMember(Principal principal) { if(groupMembers.contains(principal)) return false; if(group.equals(principal.toString())) { throw new IllegalArgumentException(); } else { groupMembers.addElement(principal); return true; } } public boolean removeMember(Principal principal) { return groupMembers.removeElement(principal); } public Enumeration members() { return groupMembers.elements(); } }</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.Principal; import java.security.acl.Group; import java.util.Enumeration; import java.util.Vector; /** * Additional class for verification Group interface */ public class GroupImpl implements Group { private Vector groupMembers; private String group; public GroupImpl(String s) { groupMembers = new Vector(50, 100); group = s; } public boolean addMember(Principal principal) { if(groupMembers.contains(principal)) return false; if(group.equals(principal.toString())) { throw new IllegalArgumentException(); } else { groupMembers.addElement(principal); return true; } } public boolean removeMember(Principal principal) { return groupMembers.removeElement(principal); } public Enumeration members() { return groupMembers.elements(); } }</pre>

GroupImpl.jad (decompiled version of Oralce GroupImpl.class) [spacing adjusted for comparison]	GroupImpl.java (Android version) [spacing adjusted for comparison]
<pre> public boolean equals(Object obj) { if(this == obj) return true; if(! (obj instanceof Group)) { return false; } else { Group group1 = (Group)obj; return group.equals(group1.toString()); } } public boolean equals(Group group1) { return equals(group1); } public String toString() { return group; } public int hashCode() { return group.hashCode(); } public boolean isMember(Principal principal) { if(groupMembers.contains(principal)) { return true; } else { Vector vector = new Vector(10); return isMemberRecurse(principal, vector); } } public String getName() { return group; } boolean isMemberRecurse(Principal principal, Vector vector) { for(Enumeration enumeration = members(); enumeration.hasMoreElements();) { boolean flag = false; Principal principal1 = (Principal)enumeration.nextElement(); if(principal1.equals(principal)) return true; if(principal1 instanceof GroupImpl) { GroupImpl groupimpl = (GroupImpl)principal1; vector.addElement(this); if(!vector.contains(groupimpl)) flag = groupimpl.isMemberRecurse(principal, vector); } else if(principal1 instanceof Group) { Group group1 = (Group)principal1; if(!vector.contains(group1)) flag = group1.isMember(principal); } if(flag) return flag; } return false; } </pre>	<pre> public boolean equals(Object obj) { if(this == obj) return true; if(! (obj instanceof Group)) { return false; } else { Group group1 = (Group)obj; return group.equals(group1.toString()); } } public boolean equals(Group group1) { return equals(group1); } public String toString() { return group; } public int hashCode() { return group.hashCode(); } public boolean isMember(Principal principal) { if(groupMembers.contains(principal)) { return true; } else { Vector vector = new Vector(10); return isMemberRecurse(principal, vector); } } public String getName() { return group; } boolean isMemberRecurse(Principal principal, Vector vector) { for(Enumeration enumeration = members(); enumeration.hasMoreElements();) { boolean flag = false; Principal principal1 = (Principal)enumeration.nextElement(); if(principal1.equals(principal)) return true; if(principal1 instanceof GroupImpl) { GroupImpl groupimpl = (GroupImpl)principal1; vector.addElement(this); if(!vector.contains(groupimpl)) flag = groupimpl.isMemberRecurse(principal, vector); } else if(principal1 instanceof Group) { Group group1 = (Group)principal1; if(!vector.contains(group1)) flag = group1.isMember(principal); } if(flag) return flag; } return false; } </pre>

GroupImpl.jad (decompiled version of Oralce GroupImpl.class) [spacing adjusted for comparison]	GroupImpl.java (Android version) [spacing adjusted for comparison]
<pre>} }</pre>	

Exhibit Copyright-N

OwnerImpl.jad (decompiled version of Oracle OwnerImpl.class) [spacing adjusted for comparison]	OwnerImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: OwnerImpl.java package sun.security.acl; import java.security.Principal; import java.security.acl.Group; import java.security.acl.LastOwnerException; import java.security.acl.NotOwnerException; import java.security.acl.Owner; import java.util.Enumeration; // Referenced classes of package sun.security.acl: // GroupImpl public class OwnerImpl implements Owner { private Group ownerGroup; public OwnerImpl(Principal principal) { ownerGroup = new GroupImpl("AclOwners"); ownerGroup.addMember(principal); } public synchronized boolean addOwner(Principal principal, Principal principal1) throws NotOwnerException { if(!isOwner(principal)) { throw new NotOwnerException(); } else { ownerGroup.addMember(principal1); return false; } } }</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.Principal; import java.security.acl.*; import java.util.Enumeration; /** * Additional class for verification Owner interface */ public class OwnerImpl implements Owner { private Group ownerGroup; public OwnerImpl(Principal principal) { ownerGroup = new GroupImpl("AclOwners"); ownerGroup.addMember(principal); } public synchronized boolean addOwner(Principal principal, Principal principal1) throws NotOwnerException { if(!isOwner(principal)) { throw new NotOwnerException(); } else { if (ownerGroup.isMember(principal1)) return false; if (!ownerGroup.isMember(principal1)) { ownerGroup.addMember(principal1); return true; } } return false; } }</pre>

OwnerImpl.jad (decompiled version of Oracle OwnerImpl.class) [spacing adjusted for comparison]	OwnerImpl.java (Android version) [spacing adjusted for comparison]
<pre> public synchronized boolean deleteOwner(Principal principal, Principal principal1) throws NotOwnerException, LastOwnerException { if(!isOwner(principal)) throw new NotOwnerException(); Enumeration enumeration = ownerGroup.members(); Object obj = enumeration.nextElement(); if(enumeration.hasMoreElements()) return ownerGroup.removeMember(principal1); else throw new LastOwnerException(); } public synchronized boolean isOwner(Principal principal) { return ownerGroup.isMember(principal); } </pre>	<pre> public synchronized boolean deleteOwner(Principal principal, Principal principal1) throws NotOwnerException, LastOwnerException { if(!isOwner(principal)) throw new NotOwnerException(); Enumeration enumeration = ownerGroup.members(); Object obj = enumeration.nextElement(); if(enumeration.hasMoreElements()) { return ownerGroup.removeMember(principal1); } else { throw new LastOwnerException(); } } public synchronized boolean isOwner(Principal principal) { return ownerGroup.isMember(principal); } </pre>

Exhibit Copyright-O

PermissionImpl.jad (decompiled version of Oracle PermissionImpl.class) [spacing adjusted for comparison]	PermissionImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: PermissionImpl.java package sun.security.acl; import java.security.acl.Permission; public class PermissionImpl implements Permission { private String permission; public PermissionImpl(String s) { permission = s; } public boolean equals(Object obj) { if(obj instanceof Permission) { Permission permission1 = (Permission)obj; return permission.equals(permission1.toString()); } else { return false; } } public String toString() { return permission; } public int hashCode() { return toString().hashCode(); } }</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.acl.Permission; /** * Additional class for verification Permission interface */ public class PermissionImpl implements Permission { private String permission; public PermissionImpl(String s) { permission = s; } public boolean equals(Object obj) { if(obj instanceof Permission) { Permission permission1 = (Permission)obj; return permission.equals(permission1.toString()); } else { return false; } } public String toString() { return permission; } /* * public int hashCode() { * return toString().hashCode(); * }*/ }</pre>

Exhibit Copyright-P

PrincipalImpl.jad (decompiled version of Oracle PrincipalImpl.class) [spacing adjusted for comparison]	PrincipalImpl.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: fieldsfirst nonlb // Source File Name: PrincipalImpl.java package sun.security.acl; import java.security.Principal; public class PrincipalImpl implements Principal { private String user; public PrincipalImpl(String s) { user = s; } public boolean equals(Object obj) { if(obj instanceof PrincipalImpl) { PrincipalImpl principalimpl = (PrincipalImpl)obj; return user.equals(principalimpl.toString()); } else { return false; } } public String toString() { return user; } public int hashCode() { return user.hashCode(); } public String getName() { return user; } }</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.Principal; /** * Additional class for verification Principal interface */ public class PrincipalImpl implements Principal { private String user; public PrincipalImpl(String s) { user = s; } public boolean equals(Object obj) { if(obj instanceof PrincipalImpl) { PrincipalImpl principalimpl = (PrincipalImpl)obj; return user.equals(principalimpl.toString()); } else { return false; } } public String toString() { return user; } public int hashCode() { return user.hashCode(); } public String getName() { return user; } }</pre>

Exhibit Copyright-Q

AclEnumerator.jad (decompiled version of Oracle AclEnumerator.class) [spacing adjusted for comparison]	AclEnumerator.java (Android version) [spacing adjusted for comparison]
<pre>// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov. // Jad home page: http://www.kpdus.com/jad.html // Decompiler options: packimports(3) fieldsfirst nonlb // Source File Name: AclImpl.java package sun.security.acl; import java.security.acl.Acl; import java.util.*; final class AclEnumerator implements Enumeration { Acl acl; Enumeration u1; Enumeration u2; Enumeration g1; Enumeration g2; AclEnumerator(Acl acl1, Hashtable hashtable, Hashtable hashtable1, Hashtable hashtable2, Hashtable hashtable3) { acl = acl1; u1 = hashtable.elements(); u2 = hashtable2.elements(); g1 = hashtable1.elements(); g2 = hashtable3.elements(); } public boolean hasMoreElements() { return u1.hasMoreElements() u2.hasMoreElements() g1.hasMoreElements() g2.hasMoreElements(); } public Object nextElement() { Acl acl1 = acl; JVM INSTR monitorenter ; if(u1.hasMoreElements()) return u1.nextElement(); if(!u2.hasMoreElements()) goto _L2; else goto _L1 _L1: u2.nextElement(); acl1; JVM INSTR monitorexit ; return;</pre>	<pre>/* * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ package org.apache.harmony.security.tests.support.acl; import java.security.acl.Acl; import java.util.*; final class AclEnumerator implements Enumeration { Acl acl; Enumeration u1; Enumeration u2; Enumeration g1; Enumeration g2; AclEnumerator(Acl acl1, Hashtable hashtable, Hashtable hashtable1, Hashtable hashtable2, Hashtable hashtable3) { acl = acl1; u1 = hashtable.elements(); u2 = hashtable2.elements(); g1 = hashtable1.elements(); g2 = hashtable3.elements(); } public boolean hasMoreElements() { return u1.hasMoreElements() u2.hasMoreElements() g1.hasMoreElements() g2.hasMoreElements(); } public Object nextElement() { Acl acl1 = acl; if(u2.hasMoreElements()) return u2.nextElement(); if(g1.hasMoreElements()) return g1.nextElement(); if(u1.hasMoreElements()) return u1.nextElement(); if(g2.hasMoreElements()) return g2.nextElement(); return acl1; } }</pre>

AclEnumerator.jad (decompiled version of Oracle AclEnumerator.class) [spacing adjusted for comparison]	AclEnumerator.java (Android version) [spacing adjusted for comparison]
<pre> _L2: _L3: if(!g1.hasMoreElements()) goto _L4; else goto _L3 g1.nextElement(); acl1; JVM INSTR monitorenter ; return; _L4: _L5: if(!g2.hasMoreElements()) goto _L6; else goto _L5 g2.nextElement(); acl1; JVM INSTR monitorenter ; return; _L6: acl1; JVM INSTR monitorenter ; goto _L7 Exception exception; exception; throw exception; _L7: throw new NoSuchElementException("Acl Enumerator"); } } </pre>	

Exhibit Copyright-R

<p>CodeSource.java (Java version)</p> <p>[spacing adjusted for comparison]</p>	<p>CodeSourceTest.java (Android version)</p> <p>[spacing adjusted for comparison]</p>
<p><u>Lines 242-59</u></p> <pre> * If this object's port (getLocation().getPort()) is not * equal to -1 (that is, if a port is specified), it must equal * <i>codesource</i>'s port. * * If this object's file (getLocation().getFile()) doesn't equal * <i>codesource</i>'s file, then the following checks are made: * If this object's file ends with "/-", * then <i>codesource</i>'s file must start with this object's * file (exclusive the trailing "-"). * If this object's file ends with a "/*", * then <i>codesource</i>'s file must start with this object's * file and must not have any further "/" separators. * If this object's file doesn't end with a "/", * then <i>codesource</i>'s file must match this object's * file with a '/' appended. * * If this object's reference (getLocation().getRef()) is * not null, it must equal <i>codesource</i>'s reference. </pre>	<p><u>Lines 598-601</u></p> <pre> /** * If this object's port (getLocation().getPort()) is not equal to -1 (that * is, if a port is specified), it must equal codesource's port. */ </pre> <p><u>Lines 629-32</u></p> <pre> /** * If this object's file (getLocation().getFile()) doesn't equal * codesource's file, then the following checks are made: ... */ </pre> <p><u>Lines 645-48</u></p> <pre> /** * ... If this object's file ends with "/-", then codesource's file must * start with this object's file (exclusive the trailing "-"). */ </pre> <p><u>Lines 667-81</u></p> <pre> /** * ... If this object's file ends with a "/*", then codesource's file must * start with this object's file and must not have any further "/" * separators. */ </pre> <p><u>Lines 693-96</u></p> <pre> /** * ... If this object's file doesn't end with a "/", then codesource's file * must match this object's file with a '/' appended. */ </pre> <p><u>Lines 711-14</u></p> <pre> /** * If this object's reference (getLocation().getRef()) is not null, it must * equal codesource's reference. */ </pre>

Exhibit Copyright-S

Excperts from CollectionCertStoreParameters.java (Java version)	Exceprts from CollectionCertStoreParameters.java (Android version)
<p><u>Lines 43-68</u></p> <pre> /** * Creates an instance of <code>CollectionCertStoreParameters</code> * which will allow certificates and CRLs to be retrieved from the * specified <code>Collection</code>. If the specified * <code>Collection</code> contains an object that is not a * <code>Certificate</code> or <code>CRL</code>, that object will be * ignored by the Collection <code>CertStore</code>. * <p> * The <code>Collection</code> is not copied. Instead, a * reference is used. This allows the caller to subsequently add or * remove <code>Certificates</code> or <code>CRL</code>s from the * <code>Collection</code>, thus changing the set of * <code>Certificates</code> or <code>CRL</code>s available to the * Collection <code>CertStore</code>. The Collection <code>CertStore</code> * will not modify the contents of the <code>Collection</code>. * <p> * If the <code>Collection</code> will be modified by one thread while * another thread is calling a method of a Collection <code>CertStore</code> * that has been initialized with this <code>Collection</code>, the * <code>Collection</code> must have fail-fast iterators. * * @param collection a <code>Collection</code> of * <code>Certificate</code>s and <code>CRL</code>s * @exception NullPointerException if <code>collection</code> is * <code>null</code> */ </pre>	<p><u>Lines 110-116</u></p> <pre> /** * Test #2 for <code>CollectionCertStoreParameters(Collection)</code> * constructor
 * Assertion: If the specified <code>Collection</code> contains an object * that is not a <code>Certificate</code> or <code>CRL</code>, that object * will be ignored by the Collection <code>CertStore</code>. */ </pre> <p><u>Lines 132-140</u></p> <pre> /** * Test #3 for <code>CollectionCertStoreParameters(Collection)</code> * constructor
 * Assertion: The Collection is not copied. Instead, a reference is used. * This allows the caller to subsequently add or remove Certificates or * CRLs from the Collection, thus changing the set of Certificates or CRLs * available to the Collection CertStore. The Collection CertStore will * not modify the contents of the Collection */ </pre>
<p><u>Lines 75-79</u></p> <pre> /** * Creates an instance of <code>CollectionCertStoreParameters</code> with * the default parameter values (an empty and immutable * <code>Collection</code>). */ </pre>	<p><u>Lines 50-54</u></p> <pre> /** * Test #1 for <code>CollectionCertStoreParameters()</code> constructor
 * Assertion: Creates an instance of CollectionCertStoreParameters * with the default parameter values (an empty and immutable Collection) */ </pre>